Syntax Pythonu

Spustenie syntaxe Python

Ako sme sa dozvedeli na predchádzajúcej stránke, syntax Pythonu sa dá spustiť zápisom priamo do príkazového riadku:

```
>>> print("Hello, World!")
Hello, World!
```

Alebo vytvorením súboru python na serveri pomocou prípony súboru .py a jeho spustením v príkazovom riadku:

```
C:\Users\Your Name>python myfile.py
```

Odsadenie Pythonu

Odsadenie sa týka medzier na začiatku riadku kódu.
Ak je v iných programovacích jazykoch odsadenie v kóde iba na čitateľnosť, odsadenie v Pythone je veľmi dôležité.
Python používa odsadenie na označenie bloku kódu.

príklad

```
if 5 > 2:
  print("Five is greater than two!")
```

Spustiť príklad »

Ak preskočíte odsadenie, Python vám poskytne chybu:

príklad

Chyba syntaxe:

```
if 5 > 2:
print("Five is greater than two!")
```

Spustiť príklad »

Počet medzier závisí od vás ako programátora, musí však byť aspoň jeden.

príklad

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

Spustiť príklad »

Musíte použiť rovnaký počet medzier v rovnakom bloku kódu, inak vám Python dá chybu:

príklad

Chyba syntaxe:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```
Spustiť príklad »

Premenné v Pythone

V Pythone sa premenné vytvoria, keď k nim priradíte hodnotu:

príklad

Premenné v Pythone:

```
x = 5
y = "Hello, World!"
```
Spustiť príklad »

Python nemá príkaz na deklarovanie premennej.

Komentáre Python

Komentáre môžu byť použité na vysvetlenie kódu Python.
Komentáre môžu byť použité na zvýšenie čitateľnosti kódu.
Komentáre môžu byť použité na zabránenie spustenia pri testovaní kódu.

Vytvorenie komentára

Komentáre začínajú #znakom a Python ich ignoruje:

príklad

```
#This is a comment
print("Hello, World!")
```
Spustiť príklad »

Komentáre môžu byť umiestnené na konci riadku a Python bude ignorovať zvyšok riadku:

príklad

```
print("Hello, World!") #This is a comment
```
Spustiť príklad »

Komentáre nemusia byť textom na vysvetlenie kódu, môžu sa tiež použiť na zabránenie Pythonu vykonať kód:

príklad

```
#print("Hello, World!")
print("Cheers, Mate!")
```
Spustiť príklad »

Komentáre s viacerými riadkami

Python v skutočnosti nemá syntax pre viac riadkové komentáre.
Ak chcete pridať viacřádkový komentár, môžete do #každého riadku vložiť znak:

príklad

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

Spustiť príklad »

Alebo nie úplne podľa plánu, môžete použiť viacriadkový reťazec.
Pretože program Python bude ignorovať reťazcové literály, ktoré nie sú priradené premennej, môžete do svojho kódu pridať reťazec s viacerými čiarami (trojité úvodzovky) a do neho vložiť komentár:

príklad

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Spustiť príklad »

Pokiaľ reťazec nie je priradený premennej, Python kód prečíta, ale potom ho ignoruje a urobíte viacřádkový komentár.

Premenné v Pythone

Vytváranie premenných

Premenné sú kontajnery na ukladanie hodnôt údajov.
Na rozdiel od iných programovacích jazykov nemá Python žiadny príkaz na deklarovanie premennej.
Premenná sa vytvorí v okamihu, keď jej prvýkrát priradíte hodnotu.

príklad

```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```
Spustiť príklad »

Premenné nemusia byť deklarované s konkrétnym typom a môžu dokonca zmeniť typ po ich nastavení.

príklad

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

```
Sally
```
Spustiť príklad »

Premenné reťazca možno deklarovať pomocou jednoduchých alebo dvojitých úvodzoviek:

príklad

```
x = "John"
# is the same as
x = 'John'
```

```
John
John
```
Spustiť príklad »

Premenné mená

Premenná môže mať krátke meno (napríklad xay) alebo popisnejšie meno (vek, meno, total_volume). Pravidlá pre premenné Pythonu:

- Názov premennej musí začínať písmenom alebo znakom podčiarknutia
- Názov premennej nemôže začínať číslom
- Názov premennej môže obsahovať iba alfanumerické znaky a znaky podčiarknutia (Az, 0-9 a _)
- Názvy premenných rozlišujú malé a veľké písmená (vek, vek a vek sú tri rôzne premenné)

Nezabudnite, že názvy premenných rozlišujú veľké a malé písmená

Priraďte hodnotu viacerým premenným

Python vám umožňuje priradiť hodnoty viacerým premenným na jednom riadku:

príklad

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

Rovnakú hodnotu môžete priradiť viacerým premenným na jednom riadku:

príklad

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

```
Orange
Orange
Orange
```

Spustiť príklad »

Výstupné premenné

Príkaz Python printsa často používa na výstup premenných.
Na kombináciu textu a premennej používa Python +znak:

príklad

```
x = "awesome"
print("Python is " + x)
```

```
Python is awesome
```

Spustiť príklad »

+Znak môžete tiež použiť na pridanie premennej k inej premennej:

príklad

```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

```
Python is awesome
```

Spustiť príklad »

Pre čísla +funguje znak ako matematický operátor:

príklad

```
x = 5
y = 10
print(x + y)
```

```
15
```

Spustiť príklad »

Ak sa pokúsite skombinovať reťazec a číslo, Python vám dá chybu:

príklad

```
x = 5
y = "John"
print(x + y)
```

```
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

Spustiť príklad »

Globálne premenné

Premenné, ktoré sa vytvárajú mimo funkcie (ako vo všetkých vyššie uvedených príkladoch), sa nazývajú globálne premenné.
Globálne premenné môžu využívať všetci, tak vo vnútri funkcií, ako aj mimo nich.

príklad

Vytvorte premennú mimo funkcie a použite ju vo funkcii

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

```
Python is awesome
```

Spustiť príklad »

Ak vo funkcii vytvoríte premennú s rovnakým názvom, táto premenná bude lokálna a dá sa použiť iba vo funkcii. Globálna premenná s rovnakým názvom zostane rovnaká, globálna as pôvodnou hodnotou.

príklad

Vytvorte premennú vo vnútri funkcie s rovnakým názvom ako globálna premenná

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

```
Python is fantastic
Python is awesome
```

Spustiť príklad »

Globálne kľúčové slovo

Normálne, keď vytvoríte premennú vo funkcii, táto premenná je lokálna a dá sa použiť iba v rámci tejto funkcie.
Na vytvorenie globálnej premennej vo funkcii môžete použiť `global` kľúčové slovo.

príklad

Ak použijete `global` kľúčové slovo, premenná patrí do globálneho rozsahu:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

```
Python is fantastic
```


Použite tiež `global`kľúčové slovo, ak chcete zmeniť globálnu premennú vo vnútri funkcie.

príklad

Ak chcete zmeniť hodnotu globálnej premennej vo funkcii, pozrite si premennú pomocou `global` kľúčového slova:

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

```
Python is fantastic
```

Dátové typy Python

Vstavané typy údajov

V programovaní je typ údajov dôležitým pojmom.
Premenné môžu ukladať údaje rôznych typov a rôzne typy môžu robiť rôzne veci.
Python má predvolene zabudované nasledujúce typy údajov v týchto kategóriách:

Typ textu:	`str`
Numerické typy:	`int`, `float`, `complex`
Typy sekvencií:	`list`, `tuple`, `range`
Typ mapovania:	`dict`

Typy súprav: `set`, `frozenset`

Booleovský typ: `bool`

Binárne typy: `bytes`, `bytearray`, `memoryview`

Získanie typu údajov

Typ funkcie ľubovoľného objektu môžete získať pomocou `type()` funkcie:

príklad :

Tlač dátového typu premennej x:

```
x = 5
print(type(x))
```

```
<class 'int'>
```

Spustiť príklad »

Nastavenie typu údajov

V Pythone je typ údajov nastavený, keď premennej priradíte hodnotu:

x = "Hello World" str

x = 20 int

x = 20.5 float

x = 1j complex

x = ["apple", "banana", "cherry"] list

x = ("apple", "banana", "cherry") tuple

x = range(6) range

x = {"name" : "John", "age" : 36} dict

x = {"apple", "banana", "cherry"} set

x = frozenset({"apple", "banana", "cherry"}) frozenset

x = True bool

x = b"Hello" bytes

x = bytearray(5) bytearray

x = memoryview(bytes(5)) memoryview

Nastavenie špecifického typu údajov

Ak chcete špecifikovať typ údajov, môžete použiť nasledujúce funkcie konštruktora:

Priklad / typ dat

x = str("Hello World") str
x = int(20) int
x = float(20.5) float
x = complex(1j) complex
x = list(("apple", "banana", "cherry")) list
x = tuple(("apple", "banana", "cherry")) tuple

x = range(6) range
x = dict(name="John", age=36) dict
x = set(("apple", "banana", "cherry")) set
x = frozenset(("apple", "banana", "cherry")) frozenset
x = bool(5) bool
x = bytes(5) bytes
x = bytearray(5) bytearray
x = memoryview(bytes(5)) memoryview

```
x = 5
print(type(x))
```

```
<class 'int'>
```

Pythonove čísla

V Pythone sú tri číselné typy:

- int
- plavajuce, pohyblive
- komplexné

Premenné číselných typov sa vytvárajú, keď im priradíte hodnotu:

príklad

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Na overenie typu ľubovoľného objektu v Pythone použite `type()`funkciu:

príklad

```
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Spustiť príklad »

Int

Int alebo celé číslo je celé číslo, kladné alebo záporné, bez desatinných miest, neobmedzenej dĺžky.

príklad

celé čísla:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Spustiť príklad »

Plavajuce, pohyblive

Float alebo „číslo s pohyblivou rádovou čiarkou“ je číslo, kladné alebo záporné, obsahujúce jedno alebo viac desatinných miest.

príklad

plaváky:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
```

```
print(type(y))
print(type(z))
```
```
<class 'float'>
<class 'float'>
<class 'float'>
```
Spustiť príklad »

Float môžu byť tiež vedecké čísla s písmenom „e“, ktoré označujú silu 10.

príklad

plaváky:

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```
```
<class 'float'>
<class 'float'>
<class 'float'>
```
Spustiť príklad »

komplexné

Komplexné čísla sa píšu s imaginárnou časťou „j“:

príklad

complex:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```
```
<class 'complex'>
<class 'complex'>
<class 'complex'
```
Spustiť príklad »

Typ Konverzia

Môžete previesť z jedného typu na iný s `int()`, `float()` a `complex()` metódy:

príklad

Prevod z jedného typu na iný:

```python
x = 1 # int
y = 2.8 # float
z = 1j # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

```
1.0
2
(1+0j)
<class 'float'> <class 'int'> <class 'complex'>
```

Spustiť príklad »

Poznámka: Komplexné čísla nemôžete prevádzať na iný typ čísla.

Náhodné číslo

Python nemá `random()` funkciu na vytváranie náhodných čísel, ale Python má vstavaný modul, `random` ktorý sa dá použiť na vytváranie náhodných čísel:

príklad

Importujte náhodný modul a zobrazte náhodné číslo od 1 do 9:

```python
import random
print(random.randrange(1,10))
```

```
8
```

Python Casting

Zadajte typ premennej

Môže sa stať, že budete chcieť špecifikovať typ premennej. To je možné urobiť pomocou casting. Python je objektovo orientovaný jazyk

a ako taký používa triedy na definovanie typov údajov vrátane ich primitívnych typov.
Odlievanie v pythone sa preto vykonáva pomocou konštruktorových funkcií:

- `int ()` - vytvára celé číslo z celočíselného literálu, floatového literálu (zaokrúhľovaním nadol na celé predchádzajúce číslo) alebo literálu reťazca (ak reťazec predstavuje celé číslo)
- `float ()` - vytvára číslo float z celočíselného literálu, floatového literálu alebo strunového literálu (za predpokladu, že reťazec predstavuje float alebo celé číslo)
- `str ()` - vytvára reťazec zo širokej škály typov údajov vrátane reťazcov, celočíselných literálov a plávajúcich literálov

príklad

celé čísla:

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

```
1
2
3
```

Spustiť príklad »

príklad

plaváky:

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

```
1.0
2.8
3.0
4.2
```

Spustiť príklad »

príklad

Strings:

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

```
s1
2
3.0
```

Reťazce Python

Literárne reťazce

Reťazce literálu v pythone sú obklopené jednoduchými úvodzovkami alebo dvojitými úvodzovkami.
„ahoj“ je to isté ako „ahoj“ .
Môžete zobraziť literál reťazcov s `print()`funkciou:

príklad

```
print("Hello")
print('Hello')
```

```
Hello
Hello
```

Spustiť príklad »

Priraďte reťazec premennej

Priradenie reťazca k premennej sa uskutoční s názvom premennej, za ktorým nasleduje rovnaké znamienko a reťazec:

príklad

```
a = "Hello"
print(a)
```

Spustiť príklad »

Multiline Strings

Viacreťazcový reťazec môžete premennej priradiť pomocou troch úvodzoviek:

príklad

Môžete použiť tri dvojité úvodzovky:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua
```

Spustiť príklad »

Alebo tri jednoduché úvodzovky:

príklad

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
```

```
ut labore et dolore magna aliqua.'''
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua
```

Spustiť príklad »

Poznámka: vo výsledku sa zalomenia riadkov vkladajú na rovnaké miesto ako v kóde.

Reťazce sú polia

Podobne ako mnoho iných populárnych programovacích jazykov, aj reťazce v Pythone sú poliami bajtov predstavujúcich znaky unicode.
Avšak Python nemá typ dátových znakov, jediný znak je jednoducho reťazec s dĺžkou 1.
Štvorcové zátvorky sa môžu použiť na prístup k prvkom reťazca.

príklad

Získajte znak na pozícii 1 (nezabudnite, že prvý znak má pozíciu 0):

```
a = "Hello, World!"
print(a[1])
```

```
e
```

Spustiť príklad »

krájanie

Pomocou syntaxe segmentu môžete vrátiť rozsah znakov.
Zadajte počiatočný index a koncový index oddelený dvojbodkou, aby ste vrátili časť reťazca.

príklad

Získajte znaky z pozície 2 do pozície 5 (nie sú súčasťou):

```
b = "Hello, World!"
print(b[2:5])
```

```
llo
```

Spustiť príklad »

Negatívne indexovanie

Na spustenie rezu od konca reťazca použite záporné indexy:

príklad

Znaky získajte od pozície 5 do pozície 1, počítanie začína od konca reťazca:

```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

Spustiť príklad »

Dĺžka reťazca

Ak chcete získať dĺžku reťazca, použite `len()`funkciu.

príklad

`len()`Vracia dĺžku reťazca:

```
a = "Hello, World!"
print(len(a))
```

13Spustiť príklad »

Metódy string

Python má sadu vstavaných metód, ktoré môžete použiť na reťazcoch.

príklad

`strip()`Metóda odstraňuje medzery od začiatku alebo na konci:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

Hello, World!Spustiť príklad »

príklad

`lower()`Metóda vracia reťazec v malými písmenami:

```
a = "Hello, World!"
print(a.lower())
```

hello, world!Spustiť príklad »

príklad

`upper()`Metóda vracia reťazec veľkými písmenami:

```
a = "Hello, World!"
print(a.upper())
```

HELLO, WORLD!Spustiť príklad »

príklad

`replace()`Metóda nahrádza reťazec iným reťazcom:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Jello, World!Spustiť príklad »

príklad

`split()`Metóda rozdeľuje reťazec do čiastkových, ak zistí, inštancií separátora:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

['Hello', ' World!']Spustiť príklad »

Ďalšie informácie o metódach String nájdete v našom referenčnom dokumente String Methods

Skontrolujte reťazec

Na kontrolu, či sa v reťazci nachádza určitá fráza alebo znak, môžeme použiť kľúčové slová `in`alebo `not in`.

príklad

Skontrolujte, či sa v nasledujúcom texte nenachádza fráza „ain“:

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x)
```

```
True
```

Spustiť príklad »

príklad

Skontrolujte, či sa v nasledujúcom texte NIE nachádza fráza „ain“:

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" not in txt
print(x)
```

Spustiť príklad »

Spojenie reťazcov

Na zreťazenie alebo kombinovanie dvoch reťazcov môžete použiť operátor +.

príklad

Zlúčiť premennú `a`s premennou `b`do premennej `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

```
HelloWorld
```

Spustiť príklad »

príklad

Ak chcete medzi nimi pridať medzeru, pridajte `" "`:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

```
Hello World
```

Spustiť príklad »

Formát reťazca

Ako sme sa dozvedeli v kapitole Python Premenné, nemôžeme takto kombinovať reťazce a čísla:

príklad

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```
```
TypeError: must be str, not int
```
Spustiť príklad »

Ale môžeme kombinovať reťazce a čísla pomocou `format()` metódy!
`format()` Metóda trvá odovzdané argumenty, formátuje ich a umiestni ich do reťazca, kde sú zástupné symboly `{}` sú:

príklad

Použite `format()` metódu na vloženie čísel do reťazcov:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```
```
My name is John, and I am 36
```
Spustiť príklad »

Metóda format () vyžaduje neobmedzený počet argumentov a umiestňuje sa do príslušných zástupných symbolov:

príklad

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```
```
I want 3 pieces of item 567 for 49.95 dollars.
```
Spustiť príklad »

Môžete použiť čísla indexov, `{0}`aby ste sa uistili, že argumenty sú umiestnené v správnych zástupných znakoch:

príklad

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```
```
I want to pay 49.95 dollars for 3 pieces of item 567
```
Spustiť príklad »

Escape Character

Ak chcete do reťazca vložiť znaky, ktoré sú nezákonné, použite znak escape.
Únikový znak je lomítko, za `\`ktorým nasleduje znak, ktorý chcete vložiť.

Príkladom nezákonného znaku je dvojitá úvodzovka vo vnútri reťazca, ktorá je obklopená dvojitými úvodzovkami:

príklad

Ak použijete dvojité úvodzovky vo vnútri reťazca, ktorý je obklopený dvojitými úvodzovkami, dostanete chybu:

```
txt = "We are the so-called "Vikings" from the north."
```

```
SyntaxError: invalid syntax
```

Spustiť príklad »

Tento problém vyriešite pomocou znaku escape \":

príklad

Únikový znak vám umožňuje použiť dvojité úvodzovky, ak to za normálnych okolností nie je dovolené:

```
txt = "We are the so-called \"Vikings\" from the north."
```

```
We are the so-called "Vikings" from the north.
```

Ostatné únikové znaky používané v Pythone:

Code	Result	Try it
\'	Jednoduchá ponuka	
\\	spätné lomítko	
\n	Nový riadok	
\r	Carriage navrat	
\t	Tabulator	
\b	medzernik	
\f	Informačný kanál	
\ooo	Octal value	
\xhh	hex hodnota	

Metódy string

Python má sadu vstavaných metód, ktoré môžete použiť na reťazcoch.
Poznámka: Všetky metódy reťazcov vracajú nové hodnoty. Nemenia pôvodný reťazec.

capitalize () Skonvertuje prvý znak na veľké písmená
casefold () Skonvertuje reťazec na malé písmená
center () Vracia centrovaný reťazec
count () Vráti, koľkokrát sa zadaná hodnota vyskytne v reťazci
encode () Vráti kódovanú verziu reťazca
endwith () Vráti true, ak reťazec končí zadanou hodnotou
expandtabs () Nastavuje veľkosť záložky reťazca
find () Vyhľadá reťazec pre zadanú hodnotu a vráti polohu, kde sa našiel
format () Naformátuje zadané hodnoty v reťazci
format_map () Naformátuje zadané hodnoty v reťazci
index () Vyhľadá reťazec pre zadanú hodnotu a vráti polohu, kde sa našiel
isalnum () Vráti hodnotu true, ak sú všetky znaky v reťazci alfanumerické
isalpha () Vráti true, ak sú všetky znaky v reťazci v abecede
isdecimal () Vráti true, ak sú všetky znaky v reťazci desatinné miesta
isdigit () Vráti true, ak sú všetky znaky v reťazci číslice
isidentifier () Vráti true, ak je reťazec identifikátor
islower () Vracia true, ak sú všetky znaky v reťazci malé
isnumeric () Vráti hodnotu true, ak sú všetky znaky v reťazci číselné
isprintable () Vráti true, ak je možné tlačiť všetky znaky v reťazci
isspace () Vráti true, ak všetky znaky v reťazci sú medzery
istitle () Vráti true, ak sa reťazec riadi pravidlami názvu
isupper () Vracia true, ak sú všetky znaky v reťazci veľké
join () Pripojí prvky iterovateľné na koniec reťazca
ljust () Vracia ľavú odôvodnenú verziu reťazca
Lower () Skonvertuje reťazec na malé písmená
lstrip () Vráti verziu reťazca vľavo
maketrans () Vracia prekladovú tabuľku, ktorá sa použije v prekladoch
partition () Vracia n-ticu, kde je reťazec rozdelený na tri časti
lace () Vracia reťazec, kde je zadaná hodnota nahradená zadanou hodnotou

rfind () Vyhľadá reťazec pre zadanú hodnotu a vráti poslednú pozíciu, kde sa našiel
rindex () Vyhľadá reťazec pre zadanú hodnotu a vráti poslednú pozíciu, kde sa našiel
rjust () Vracia správne zarovnanú verziu reťazca
rpartition () Vráti n-ticu, kde je reťazec rozdelený na tri časti
rsplit () Rozdeľuje reťazec v určenom oddeľovači a vracia zoznam
rstrip () Vráti verziu reťazca s pravým orezom
split () Rozdeľuje reťazec v určenom oddeľovači a vracia zoznam
splitlines () Rozdeľuje reťazec pri zalomení riadku a vracia zoznam
beginwith () Vráti true, ak sa reťazec začína zadanou hodnotou
strip () Vracia orezanú verziu reťazca
swapcase () Swapcase, malé písmená sa stávajú veľké a naopak
title () Skonvertuje prvý znak každého slova na veľké písmená
translate () Vráti preložený reťazec
upper () Skonvertuje reťazec na veľké písmená
zfill () Na začiatok vyplní reťazec zadaným počtom 0 hodnôt

Python Booleans

Booleans predstavujú jednu z dvoch hodnôt: `True` alebo `False`.

Booleovské hodnoty

Pri programovaní často potrebujete vedieť, či je výraz výraz `True` alebo `False`.
Môžete vyhodnotiť akýkoľvek výraz v Pythone a získať jednu z dvoch odpovedí, `True` alebo `False`.
Keď porovnáte dve hodnoty, vyhodnotí sa výraz a Python vráti booleovskú odpoveď:

príklad

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

Keď spustíte podmienku v príkaze if, Python sa vráti `True` alebo `False`:

príklad

Vytlačte správu na základe toho, či je podmienka `True` alebo `False`:

```
a = 200
b = 33

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
b is not greater than a
```

Vyhodnoťte hodnoty a premenné

Táto `bool()`funkcia vám umožňuje vyhodnotiť akúkoľvek hodnotu a poskytnúť vám `True`alebo `False` na oplátku,

príklad

Vyhodnoťte reťazec a číslo:

```
print(bool("Hello"))
print(bool(15))
```

```
True
True
```

príklad

Vyhodnoťte dve premenné:

```
x = "Hello"
y = 15

print(bool(x))
print(bool(y))
```

```
True
True
```

Väčšina hodnôt je pravda

Hodnotí sa takmer akákoľvek hodnota, `True`ak má nejaký obsah.
Akýkoľvek reťazec je `True`, okrem prázdnych reťazcov.
Akékoľvek číslo `True` okrem `0`.
Akýkoľvek zoznam, n-tica, množina a slovník sú `True` okrem prázdnych.

príklad

Nasledujúce vráti hodnotu True:

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

```
True
True
True
```

Niektoré hodnoty sú nepravdivé

V skutočnosti, nie je veľa hodnôt, ktorého výsledkom `False`, s výnimkou prázdnych hodnôt, ako je `()`, `[]`, `{}`, `""`, počet `0` a hodnota `None`. Hodnota sa samozrejme `False` hodnotí `False`.

príklad

Nasledujúce vráti False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

```
False
False
False
False
False
False
False
```

Jedna hodnota alebo objekt v tomto prípade sa vyhodnotí `False` a to je, ak máte objekty, ktoré sú vyrobené z triedy s `__len__` funkciou, ktorá sa vracia `0` alebo `False`:

príklad

```
class myclass():
  def __len__(self):
    return 0

myobj = myclass()
print(bool(myobj))
```

```
False
```

Funkcie môžu vrátiť booleovský výraz

Python má tiež veľa vstavaných funkcií, ktoré vracajú booleovskú hodnotu, napríklad `isinstance()` funkciu, ktorá sa dá použiť na určenie, či je objekt určitého typu údajov:

príklad

Skontrolujte, či je objekt celé alebo nie:

```
x = 200
print(isinstance(x, int))
```
```
True
```

Operátori Pythonu

Operátory sa používajú na vykonávanie operácií s premennými a hodnotami.

Python rozdeľuje operátorov do nasledujúcich skupín:

- Aritmetické operátory
- Prevádzkovatelia priradenia
- Porovnávacie subjekty
- Logické operátory
- Prevádzkovatelia identity
- Členskí operátori
- Bitové operátory

Aritmetické operátory Python

Aritmetické operátory sa používajú s číselnými hodnotami na vykonávanie bežných matematických operácií:

Príklad názvu operátora Vyskúšajte to

+ Sčítanie x + y

- Odčítanie x - y

***** Násobenie x * y

/ Delenie x / y

% Modul x% y

```
x = 5
y = 2
print(x % y)
```
```
1
```

** umocňovanie x ** y

```
x = 2
```

```
y = 5
print(x ** y)     #to iste ako 2*2*2*2*2
32
```

// Delenie floor x // y

```
x = 15
y = 2

print (x // y)
7

# delenie floor // zaokrúhľuje výsledok na najbližšie celé číslo
```

Operátori priradenia Pythonu

Operátory priradenia sa používajú na priradenie hodnôt k premenným:

Operator	priklad	to iste ako
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
***=**	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

```
x = 5
x%=3
print(x)
2
```

//=	x //= 3	x = x // 3

```
x = 5
x//=3
```

```
print(x)
```

```
1
```

**= x **= 3 x = x ** 3

```
x = 5
x **= 3
print(x)
```

```
125
```

&= x &= 3 x = x & 3

```
x = 5
x &= 3
print(x)
```

```
1
```

|= x |= 3 x = x | 3

```
x = 5
x |= 3
print(x)
```

```
7
```

^= x ^= 3 x = x ^ 3

```
x = 5
x ^= 3
print(x)
```

```
6
```

>>= x >>= 3 x = x >> 3

```
x = 5
x >>= 3
print(x)
```

```
0
```

<<= x <<= 3 x = x << 3

```
x = 5
x <<= 3
print(x)
```

```
40
```

Porovnávacie operátory Pythonu

Porovnávacie operátory sa používajú na porovnanie dvoch hodnôt:

== Rovnaké x == y

! = Nerovná sa x! = Y

> Väčšie ako x> y

< Menej ako x <y

> = Väčšie alebo rovné x> = y

<= Menšie alebo rovné x <= y y

Logické operátory Pythonu

Logické operátory sa používajú na kombinovanie podmienených príkazov:

and Vráti hodnotu true, ak sú obidve príkazy pravdivé x <5 a x <10

or Vráti hodnotu true, ak je jedno z tvrdení pravdivé x <5 alebo x <4

not Obrátiť výsledok, vráti Falošné, ak výsledok nie je pravdivý not(x <5 and x <10)

Python Identity Operators

Operátory identity sa používajú na porovnávanie objektov, nie ak sú rovnaké, ale ak sú skutočne rovnakým objektom, s rovnakým umiestnením pamäte:

is Vráti true, ak sú obe premenné rovnaký objekt x je y

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
```

```
print(x is z)
# vráti True, pretože z je rovnaký objekt ako x
print(x is y)
#vráti False, pretože x nie je ten istý objekt ako y, aj keď majú rovnaký obsah
print(x == y 
#na preukázanie rozdielu medzi „je“ a „==“: toto porovnanie vráti hodnotu true, pretože x sa rovná y

True
False
True
```

to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y

is not Vráti true, ak obidve premenné nie sú ten istý objekt x nie je y

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is not z)
# vráti False, pretože z je rovnaký objekt ako x
print (x is not y)
# vráti true, pretože x nie je rovnaký objekt ako y, aj keď majú rovnaký obsah
print (x! = y)
# na preukázanie rozdielu medzi „nie je“ a „! =“: toto porovnanie vráti hodnotu False, pretože x sa rovná
False
True
False
```

Operátori členstva v Pythone

Operátori členstva sa používajú na testovanie, či je sekvencia prezentovaná v objekte:

in Vráti hodnotu True, ak je v objekte x in y prítomná sekvencia so zadanou hodnotou

```
x = ["apple", "banana"]
print("banana" in x)
# vráti Pravda, pretože v zozname je uvedená sekvencia s hodnotou „banán“
True
```

not in Vráti True, ak sekvencia so zadanou hodnotou nie je prítomná v objekte

```
x = ["apple", "banana"]
print("pineapple" not in x)
# vráti True, pretože sekvencia s hodnotou „ananás“ nie je v liste
```

Operátory Python Bitwise

Na porovnávanie (binárnych) čísel sa používajú bitové operátory:

& *AND* Nastaví každý bit na 1, ak sú oba bity 1

| *ALEBO* Nastavuje každý bit na 1, ak jeden z dvoch bitov je 1

^ *XOR* Nastaví každý bit na 1, ak je iba jeden z dvoch bitov 1

~ *NOT* Invertuje všetky bity

<< *Nulová výplň doľava* Posunutím doľava vyplni nuly sprava a bity posunie do lava

>> Už hore vyssie podpísaný posun doprava .

Zoznamy Pythonu

Kolekcie Pythonu (polia)

V programovacom jazyku Python existujú štyri typy údajov kolekcie:

- **Zoznam** je zbierka, ktorá je usporiadaná a meniteľná. Umožňuje duplikovať členov.

- **Tuple** je kolekcia, ktorá je usporiadaná a nemenná. Umožňuje duplikovať členov.
- **Sada** je kolekcia, ktorá nie je usporiadaná a nie je indexovaná. Žiadni duplicitní členovia.
- **Slovník** je kolekcia, ktorá nie je usporiadaná, meniteľná a indexovaná. Žiadni duplicitní členovia.

Pri výbere typu kolekcie je užitočné porozumieť vlastnostiam tohto typu. Výber správneho typu pre konkrétny súbor údajov by mohol znamenať zachovanie významu a mohlo by to znamenať zvýšenie efektívnosti alebo bezpečnosti.

zoznam

Zoznam je zbierka, ktorá je usporiadaná a meniteľná. V Pythone sú zoznamy napísané hranatými zátvorkami.

príklad

Vytvorenie zoznamu:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

Prístup k položkám

K položkám zoznamu sa dostanete pomocou indexového čísla:

príklad

Vytlačte druhú položku zoznamu:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
banana
```

Negatívne indexovanie

Záporné indexovanie znamená od začiatku, `-1` odkazuje na poslednú položku, `-2` odkazuje na druhú poslednú položku atď.

príklad

Vytlačí poslednú položku zoznamu:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```
```
cherry
```

Rozsah indexov

Rozsah indexov môžete určiť zadaním, kde sa má začať a kde sa má ukončiť rozsah.
Pri určovaní rozsahu bude návratovou hodnotou nový zoznam so zadanými položkami.

príklad

Vráťte tretiu, štvrtú a piatu položku:

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon",
 "mango"]
print(thislist[2:5])
```
```
['cherry', 'orange', 'kiwi']
```

Poznámka: Vyhľadávanie sa začne indexom 2 (vrátane) a končí indexom 5 (nie je súčasťou).
Pamätajte, že prvá položka má index 0.

Ak vynecháte počiatočnú hodnotu, rozsah sa začne pri prvej položke:

príklad

Tento príklad vráti položky zo začiatku na „oranžové“:

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon",
 "mango"]
print(thislist[:4])
```
```
['apple', 'banana', 'cherry', 'orange']
```

Ak vynecháte koncovú hodnotu, rozsah prejde na koniec zoznamu:

príklad

Tento príklad vráti položky z položky „cherry“ a na koniec:

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon",
```

```
 "mango"]
print(thislist[2:])
```

```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

Rozsah negatívnych indexov

Ak chcete začať vyhľadávanie od konca zoznamu, zadajte záporné indexy:

príklad

Tento príklad vráti položky z indexu -4 (vrátane) na index -1 (vylúčené)

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon",
 "mango"]
print(thislist[-4:-1])
```

```
['orange', 'kiwi', 'melon']
```

Zmeňte hodnotu položky

Ak chcete zmeniť hodnotu konkrétnej položky, pozrite si číslo indexu:

príklad

Zmeniť druhú položku:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

Slučka cez zoznam

Prostredníctvom slučky môžete prechádzať položkami zoznamu `for` :

príklad

Vytlačte všetky položky v zozname, jeden po druhom:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

```
apple
banana
cherry
```

Viac o `for` slučkách sa dozviete v našej kapitole Python For Loops .

Skontrolujte, či položka existuje

Ak chcete zistiť, či je určitá položka v zozname, použite `in`kľúčové slovo:

príklad

Skontrolujte, či je v zozname uvedené „jablko“:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

```
Yes, 'apple' is in the fruits list
```

Dĺžka zoznamu

Ak chcete zistiť, koľko položiek má zoznam, použite `len()` funkciu:

príklad

Vytlačte počet položiek v zozname:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
3
```

Pridať položky

Ak chcete pridať položku na koniec zoznamu, použite metódu `append ()` :

príklad

Použitie `append()`metódy na pripojenie položky:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
['apple', 'banana', 'cherry', 'orange']
```

Ak chcete pridať položku do určeného indexu, použite metódu `insert ()` :

príklad

Vložte položku ako druhú pozíciu:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
['apple', 'orange', 'banana', 'cherry']
```

Odstrániť položku

Existuje niekoľko metód na odstránenie položiek zo zoznamu:

príklad

`remove()` Metóda Odstráni určenú položku:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
['apple', 'cherry']
```

príklad

`pop()` Metóda odstráni určenom indexu (alebo poslednú položku, pokiaľ nie je zadaný index):

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
['apple', 'banana']
```

príklad

`del`Kľúčové slovo odstraňuje určenom indexu:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

príklad

`del` Kľúčové slovo môže zoznam tiež úplne odstrániť:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

```
['banana', 'cherry']
```

príklad

`clear()` Metóda vyprázdni zoznam:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

Skopírujte zoznam

Nemožno kopírovať zoznam jednoducho tým, že písanie `list2 = list1`, pretože: `list2` bude iba *odkaz* na `list1`, a zmeny vykonané v `list1` automaticky tiež v `list2`.

Existujú spôsoby, ako vytvoriť kópiu, jedným zo spôsobov je použitie vstavanej metódy zoznamu `copy()`.

príklad

Vytvorte kópiu zoznamu pomocou `copy()` metódy:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

Ďalším spôsobom, ako vytvoriť kópiu, je použitie vstavanej metódy `list()`.

príklad

Vytvorte kópiu zoznamu pomocou `list()` metódy:

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```
```
['apple', 'banana', 'cherry']
```

Pripojte sa k dvom zoznamom

Existuje niekoľko spôsobov, ako sa pripojiť alebo zreťaziť dva alebo viac zoznamov v Pythone.
Jedným z najjednoduchších spôsobov je použitie `+` operátora.

príklad

Pripojiť sa k dvom zoznamom:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```
```
['a', 'b', 'c', 1, 2, 3]
```

Ďalším spôsobom, ako sa spojiť s dvoma zoznamami, je pripojenie všetkých položiek zo zoznamu2 do zoznamu1, jedna po druhej:

príklad

Pripojiť list2 do zoznamu1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)
```
```
['a', 'b', 'c', 1, 2, 3]
```

Alebo môžete použiť `extend()` metódu, ktorej účelom je pridať prvky z jedného zoznamu do druhého:

príklad

Pomocou `extend()` metódy pridajte zoznam2 na koniec zoznamu1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
```
```
['a', 'b', 'c', 1, 2, 3]
```

Zoznam () Konštruktor

Na vytvorenie nového zoznamu je tiež možné použiť konštruktor `list ()` .

príklad

Použitie `list()`konštruktora na vytvorenie zoznamu:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```
```
['apple', 'banana', 'cherry']
```

Metódy zoznamu

Python má sadu vstavaných metód, ktoré môžete použiť v zoznamoch.

append() Pridá prvok na koniec zoznamu

clear() Odstráni všetky prvky zo zoznamu

copy() Vráti kópiu zoznamu

count() Vráti počet prvkov so zadanou hodnotou

extend() Na koniec aktuálneho zoznamu prida prvky zoznamu (alebo iné iterovateľné položky)

index() Vracia index prvého prvku so zadanou hodnotou

insert() Pridá prvok na určené miesto

pop() Odstráni prvok na určenej pozícii

remove() Odstráni položku so zadanou hodnotou

reverse() Obráti poradie zoznamu

sort() Zoraďuje zoznam

Pythonové zväzky

Tuple, n-tice

N-tica je kolekcia, ktorá je usporiadaná a **nemenná** . V Pythone sú n-tice napísané s okrúhlymi zátvorkami.

príklad

Vytvorenie tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Prístup k položkám Tuple

K bodovým položkám môžete pristupovať podľa indexového čísla v hranatých zátvorkách:

príklad

Vytlačte druhú položku v zväzku:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

Negatívne indexovanie

Záporné indexovanie znamená od začiatku, `-1` odkazuje na poslednú položku, `-2`odkazuje na druhú poslednú položku atď.

príklad

Vytlačte poslednú položku v zväzku:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Rozsah indexov

Rozsah indexov môžete určiť zadaním, kde sa má začať a kde sa má ukončiť rozsah.

Pri určovaní rozsahu bude návratovou hodnotou nové n-tice so zadanými položkami.

príklad

Vráťte tretiu, štvrtú a piatu položku:

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon",
 "mango")
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

Poznámka: Vyhľadávanie sa začne indexom 2 (vrátane) a končí indexom 5 (nie je súčasťou).
Pamätajte, že prvá položka má index 0.

Rozsah negatívnych indexov

Ak chcete spustiť vyhľadávanie od konca n-tice, zadajte záporné indexy:

príklad

Tento príklad vráti položky z indexu -4 (vrátane) na index -1 (vylúčené)

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon",
 "mango")
print(thistuple[-4:-1])
```

```
('orange', 'kiwi', 'melon')
```

Zmeňte hodnoty v n-tice

Po vytvorení n-tice už nemôžete meniť jej hodnoty. Trice sú **nemenné** , alebo **nemenné,** ako sa tiež nazýva.
Existuje však riešenie. Môžete konvertovať n-tice do zoznamu, meniť zoznam a zoznam konvertovať späť na n-tice.

príklad

Zmeňte n-ticu na zoznam, aby ste ju mohli zmeniť:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

```
("apple", "kiwi", "cherry")
```

Slučka cez tupl

Prostredníctvom slučky môžete prechádzať cez n-tice `for`.

príklad

Prechádzajte položkami a tlačte hodnoty:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
apple
banana
cherry
```

Viac o `for` slučkách sa dozviete v našej kapitole Python For Loops .

Skontrolujte, či položka existuje

Ak chcete zistiť, či je zadaná položka v n-tice, použite `in`kľúčové slovo:

príklad

Skontrolujte, či v tuple nie je „jablko“:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

Dlzka tuple.

Ak chcete zistiť, koľko položiek má n-tica, použite túto `len()`metódu:

príklad

Vytlačte počet položiek v zväzku:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

Pridať položky

Po vytvorení n-tice už do nej nemôžete pridať položky. Trice sú **nemenné** .

príklad

Do n-tice nemôžete pridať položky:

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

```
Traceback (most recent call last):
  File "demo_tuple_add.py", line 2, in <module>
    thistuple[3] = "orange" # This will raise an error
TypeError: 'tuple' object does not support item
assignment
```

Vytvorte tupl s jednou položkou

Ak chcete vytvoriť n-ticu iba s jednou položkou, za položku musíte pridať čiarku, pokiaľ Python nerozpozná premennú ako n-ticu.

príklad

Jedna položka je roztrhaná, nezabudnite na kommu:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

Odstrániť položky

Poznámka: Položky v n-tici nie je možné odstrániť.

Tuples sú **nemenné** , takže z nich nemôžete odstrániť položky, ale tuple môžete úplne odstrániť:

príklad

`del`Kľúčové slovo môže n-tica úplne odstrániť:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

```
NameError: name 'thistuple' is not defined
```

Pripojte sa k dvom Tuplesom

Ak sa chcete spojiť s dvoma alebo viacerými n-ticami, môžete použiť `+` operátor:

príklad

Pripojte sa k dvom n-ticám:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Konštruktor n-tice ()

Je tiež možné použiť konštruktor `tuple ()` na vytvorenie tuple.

príklad

Pomocou metódy tuple () vytvorte n-ticu:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
('apple', 'banana', 'cherry')
```

Tuple metódy

Python má dve vstavané metódy, ktoré môžete použiť na n-tice.

count () Vráti, koľkokrát sa zadaná hodnota vyskytuje v n-tici

index () Vyhľadá n-ticu pre zadanú hodnotu a vráti polohu, kde bola nájdená

Sady Pythonu

sada

Súbor je kolekcia, ktorá nie je usporiadaná a nie je indexovaná. V Pythone sú súbory písané so zloženými zátvorkami.

príklad

Vytvorenie sady:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
{'cherry', 'apple', 'banana'}
```

Poznámka: Množiny sú neusporiadané, takže si nemôžete byť istí, v akom poradí sa položky zobrazia.

Prístup k položkám

K položkám v množine nemôžete pristupovať odkazom na index, pretože sady sú neusporiadané, položky nemajú index.

Môžete však prepínať medzi položkami množiny pomocou `for` slučky alebo pomocou `in`kľúčového slova sa opýtať, či sa v množine nachádza určitá hodnota .

príklad

Prejdite cez súpravu a vytlačte hodnoty:

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

```
apple
banana
cherry
```

príklad

Skontrolujte, či sa v banke nachádza „banán“:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

Zmeniť položky

Po vytvorení nie je možné meniť jeho položky, ale môžete pridávať nové položky.

Pridať položky

Na pridanie jednej položky do sady použite `add()` metódu.
Ak chcete do sady pridať viac ako jednu položku, použite `update()` metódu.

príklad

Pridajte položku do sady pomocou tejto `add()` metódy:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'orange', 'banana', 'cherry', 'apple'}
```

príklad

Pridajte do sady viac položiek pomocou tejto `update()` metódy:

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

```
{'apple', 'grapes', 'mango', 'banana', 'orange', 'cherry'}
```

Získajte dĺžku sady

Ak chcete zistiť, koľko položiek má množina, použite túto `len()` metódu.

príklad

Získajte počet položiek v sade:

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

```
3
```

Odstrániť položku

Na odstránenie položky zo sady použite metódu `remove()`alebo `discard()`.

príklad

Odstráňte banán pomocou `remove()` metódy:

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

```
{'apple', 'cherry'}
```

Poznámka: Ak položka, ktorú chcete odstrániť, neexistuje, `remove()`spôsobí chybu.

príklad

Odstráňte banán pomocou `discard()` metódy:

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

```
{'cherry', 'apple'}
```

Poznámka: Ak je položku odstrániť, neexistuje, `discard()`bude **NOT** zvýšiť chybu.

Na `pop()`odstránenie položky môžete použiť aj metódu,, ale táto metóda odstráni *poslednú* položku. Nezabudnite, že množiny sú neusporiadané, takže nebudete vedieť, ktorá položka sa odstráni.
Vrátená hodnota `pop()`metódy je odstránená položka.

príklad

Odstráňte poslednú položku pomocou `pop()` metódy:

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

```
apple
{'cherry', 'banana'}
```

Poznámka: Množiny sú *neusporiadané* , takže keď používate `pop()`metódu, nebudete vedieť, ktorá položka sa odstráni.

príklad

`clear()` Metóda vyprázdni set:

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

```
set()
```

príklad

`del` Kľúčové slovo vymaže sadu kompletne:

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

```
NameError: name 'thisset' is not defined
```

Pripojte sa k dvom sadám

Existuje niekoľko spôsobov, ako sa pripojiť k dvom alebo viacerým súborom v Pythone.

Môžete použiť `union()` metódu, ktorá vracia novú množinu obsahujúcu všetky položky z oboch sád, alebo `update()` metódu, ktorá vkladá všetky položky z jednej sady do druhej:

príklad

`union()` Metóda vracia nový súbor so všetkými predmetmi z oboch skupín:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

```
{2, 1, 'b', 3, 'a', 'c'}
```

príklad

`update()`Metóda vloží položky v MNOŽINY2 do SET1

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

```
{'c', 1, 'a', 2, 'b', 3}
```

Poznámka: Oboje `union()` a `update()` vylúči všetky duplicitné položky.

Existujú aj iné metódy, ktoré spájajú dve sady a uchovávajú IBA duplikáty alebo NIKDY duplikáty. Skontrolujte úplný zoznam metód nastavenia v dolnej časti tejto stránky.

Sada () Konštruktor

Na vytvorenie `množiny` je tiež možné použiť konštruktor `set()` .

príklad

Pomocou konštruktora set () vytvorte množinu:

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

```
{'apple', 'cherry', 'banana'}
```

Metódy stanovenia

Python má sadu vstavaných metód, ktoré môžete použiť v množinách.

add() Pridá prvok do množiny

clear() Odstráni všetky prvky zo sady

copy() Vracia kópiu sady

difference() Vracia množinu obsahujúcu rozdiel medzi dvoma alebo viacerými množinami

differencel_update() Odstráni položky v tejto množine, ktoré sú tiež zahrnuté v inej špecifikovanej množine

discard() Odstrániť uvedenú položku

intersection() Vracia množinu, ktorá je priesečníkom dvoch ďalších množín

intersection_update() Odstráni položky v tejto množine, ktoré sa nenachádzajú v iných špecifikovaných množinách.

isdisjoint() Vracia, či dve sady majú priesečník alebo nie

issubset() Vracia, či táto sada obsahuje alebo nie

issuperset() Vracia, či táto sada obsahuje inú množinu alebo nie

pop() Odstráni prvok zo sady

remove() Odstráni určený prvok

symmetric_difference() Vracia množinu so symetrickými rozdielmi dvoch množín

symmetric_difference_update() vkladá symetrické rozdiely z tejto množiny a ďalších

union() Vráti množinu obsahujúcu spojenie množín

update() Aktualizujte súbor spojením tejto množiny a ďalších

Python slovníky

slovník

Slovník je kolekcia, ktorá nie je usporiadaná, meniteľná a indexovaná. V Pythone sú slovníky písané s zloženými zátvorkami a majú kľúče a hodnoty.

príklad

Vytvorenie a tlač slovníka:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Prístup k položkám

K položkám slovníka môžete pristupovať podľa názvu kľúča v hranatých zátvorkách:

príklad

Získajte hodnotu kľúča „model“:

```
x = thisdict["model"]
```

```
Mustang
```

Nazýva sa aj metóda, `get()`ktorá vám dá rovnaký výsledok:

príklad

Získajte hodnotu kľúča „model“:

```
x = thisdict.get("model")
```

```
Mustang
```

Zmeniť hodnoty

Hodnotu konkrétnej položky môžete zmeniť podľa jej názvu kľúča:

príklad

Zmeňte „rok“ na 2018:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Slučka cez slovník

Pomocou slučky môžete prechádzať cez slovník `for`.

Pri opakovaní v slovníku sú návratové hodnoty *kľúčmi* slovníka, existujú však aj metódy na vrátenie týchto *hodnôt* .

príklad

Vytlačte všetky názvy kľúčov v slovníku po jednom:

```
for x in thisdict:
  print(x)
```

```
brand
model
year
```

príklad

Vytlačte všetky *hodnoty* v slovníku, postupne:

```
for x in thisdict:
  print(thisdict[x])
```

```
Ford
Mustang
1964
```

príklad

Túto `values()`funkciu môžete použiť aj na vrátenie hodnôt zo slovníka:

```
for x in thisdict.values():
  print(x)
```

```
Ford
Mustang
1964
```

príklad

Slučka cez *klávesy* a *hodnoty* pomocou `items()`funkcie:

```
for x, y in thisdict.items():
  print(x, y)
```

```
brand Ford
model Mustang
year 1964
```

Skontrolujte, či existuje kľúč

Ak chcete zistiť, či je zadaný kľúč v slovníku, použite `in`kľúčové slovo:

príklad

Skontrolujte, či je v slovníku „model“:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

Dĺžka slovníka

Na určenie toho, koľko položiek (páry kľúč - hodnota) má slovník, použite túto `len()` metódu.

príklad

Vytlačte počet položiek v slovníku:

```
print(len(thisdict))
```

```
3
```

Pridávanie položiek

Pridanie položky do slovníka sa uskutoční pomocou nového kľúča indexu a priradeniu hodnoty k nemu:

príklad

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Odstránenie položiek

Existuje niekoľko metód na odstránenie položiek zo slovníka:

príklad

`pop()`Metóda odstráni položku s daným názvom kľúča:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

príklad

Táto `popitem()` metóda odstráni poslednú vloženú položku (vo verziách pred 3.7 sa namiesto nej odstráni náhodná položka):

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

príklad

`del`Kľúčové slovo odoberie položku s daným názvom kľúča:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

príklad

`del`Kľúčové slovo môže slovníka tiež úplne odstrániť:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because
"thisdict" no longer exists.
```

```
NameError: name 'thisdict' is not defined
```

príklad

`clear()`Kľúčové slovo vyprázdni slovník:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
{}
```

Skopírujte slovník

Nemôžete kopírovať slovník jednoducho tak, že napíšete `dict2 = dict1`, pretože: `dict2`bude iba *odkaz* na `dict1`a vykonané zmeny `dict1`sa automaticky vykonajú aj v `dict2`.

Existujú spôsoby, ako vytvoriť kópiu, jedným zo spôsobov je použitie vstavanej metódy slovníka `copy()`.

príklad

Vytvorte kópiu slovníka pomocou `copy()`metódy:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
```

```
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Ďalším spôsobom, ako vytvoriť kópiu, je použitie vstavanej metódy `dict()`.

príklad

Vytvorte kópiu slovníka pomocou `dict()`metódy:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Vnorené slovníky

Slovník môže obsahovať aj veľa slovníkov, nazýva sa to vnorené slovníky.

príklad

Vytvorte slovník, ktorý obsahuje tri slovníky:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
```

```
  }
}
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

Alebo ak chcete vnoriť tri slovníky, ktoré už existujú ako slovníky:

príklad

Vytvorte tri slovníky, potom vytvorte jeden slovník, ktorý bude obsahovať ďalšie tri slovníky:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

Diktátor Konštruktor

Na vytvorenie nového slovníka je tiež možné použiť konštruktor `dict ()` :

príklad

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the
#assignment
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Metódy slovníka

Python má sadu vstavaných metód, ktoré môžete použiť v slovníkoch.

clear() Odstráni všetky prvky zo slovníka

copy() Vráti kópiu slovníka

fromkeys() Vracia slovník so zadanými kľúčmi a hodnotami

get() Vracia hodnotu zadaného kľúča

items() Vracia zoznam obsahujúci n-ticu pre každý pár hodnôt kľúčov

keys() Vracia zoznam obsahujúci kľúče slovníka

pop() Odstráni prvok zadaným kľúčom

popitem() Odstráni posledný vložený pár kľúč - hodnota

setdefault() Vracia hodnotu zadaného kľúča. Ak kľúč neexistuje: vložte kľúč so zadanou hodnotou

update() Aktualizuje slovník pomocou zadaných párov kľúč - hodnota

values() Vráti zoznam všetkých hodnôt v slovníku

Python If ... Else

Príkazy Python If

Python podporuje obvyklé logické podmienky z matematiky:

- Rovná sa: `a == b`

- Nie sa rovná: `a! = B`
- Menej ako: `a <b`
- Menej alebo sa rovná: `a <= b`
- Väčšie ako: `a> b`
- Väčšie alebo sa rovná: `a> = b`

Tieto podmienky môžu byť použité niekoľkými spôsobmi, najčastejšie v príkazoch if if a loop.
Príkaz if sa píše pomocou kľúčového slova `if` .

príklad

Ak vyhlásenie:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

V tomto príklade budeme používať dve premenné, a `b` , ktoré sa používajú ako súčasť if otestovať, či `b` je väčšia ako . Pretože `a` je `33` a `b` je `200` , vieme, že 200 je väčšie ako 33, a preto tlačíme na obrazovku, že „b je väčšie ako a".

odsadenie

Python sa pri definovaní rozsahu v kóde spolieha na odsadenie (medzeru na začiatku riadku). Iné programovacie jazyky na tento účel často používajú zložené zátvorky.

príklad

Ak príkaz, bez odsadenia (spôsobí chybu):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

```
  File "demo_if_error.py", line 4
    print("b is greater than a")
        ^
IndentationError: expected an indented block
```

Elif

`Elif` kľúčové slovo je Pythons spôsob, ako hovoriť "v prípade, že predchádzajúce podmienky neboli pravdivé, potom skúste túto podmienku".

príklad

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a and b are equal
```

V tomto príklade `a` je rovná `b` , takže prvá podmienka nie je pravdivá, ale podmienka `elif` je pravdivá, takže tlačíme na obrazovku, že „a a b sú rovnaké".

inak

`Else` kľúčových úlovky niečo, ktorý nie je zachytený z predchádzajúcich podmienok.

príklad

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

V tomto príklade je väčší ako `b` , takže prvá podmienka nie je pravda, aj `elif` podmienka nie je pravda, takže ideme do `iného` stavu a vytlačiť na obrazovku, že "je väčší ako b".

Môžete tiež mať `else`bez `elif`:

príklad

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
b is not greater than a
```

Krátka ruka If

Ak chcete vykonať iba jeden príkaz, môžete ho umiestniť na rovnaký riadok ako príkaz if.

príklad

Jeden riadok, ak príkaz:

```
if a > b: print("a is greater than b")
```

```
"a is greater than b"
```

Krátka ruka, ak ... Inak

Ak chcete vykonať iba jeden príkaz, jeden pre ak a jeden pre iné, môžete ho umiestniť na jeden riadok:

príklad

Jeden riadok, ak inde:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

```
B
```

Na rovnakom riadku môžete mať aj niekoľko ďalších príkazov:

príklad

Jeden riadok, ak inde, s 3 podmienkami:

```
a = 330
b = 330
print("A") if a > b else print("=") if a ==
b else print("B")
```

=

a

A kľúčové slovo je logický operátor a slúži k kombinovať podmienené príkazy:

príklad

Otestujte, či a je väčší ako b a či c je väčší ako a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```

alebo

or Kľúčové slovo je logický operátor a slúži k kombinovať podmienené príkazy:

príklad

Test, či a je väčší ako b, ALEBO ak a je väčší ako c:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

Vnorené, ak

Vo `if` výpisoch môžete mať príkazy `if`, ktoré sa nazývajú *vnorené* `if` príkazy.

príklad

```
x = 41
if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
Above ten,
and also above 20!
```

Úspešné vyhlásenie

`if` príkazy nemôžu byť prázdne, ale ak z nejakého dôvodu máte `if` vyhlásenie bez obsahu, do `pass` príkazu vložte príkaz, aby ste sa vyhli chybe.

príklad

```
a = 33
b = 200
if b > a:
  pass
# má prázdny príkaz ako je tento. Spôsobil by chybu bez príkazu pass
```

Python, while "Pokiaľ čo"

"Pokiaľ čo" Python

Python má dva príkazy primitívnej slučky:

- `while` slučka
- `for` slučka

While slučka

S `while` slučky, môžeme vykonať sadu príkazov tak dlho, kým je podmienka pravdivá.

príklad

Tlač i, ak je i menšie ako 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

Poznámka: nezabudnite zvyšovať i, inak bude slučka pokračovať navždy.

`While` slučka vyžaduje relevantné premenné, aby bola pripravená v tomto prípade musíme definovať indexovacie premennú, `i` , ktorú sme nastavená na hodnotu 1.

Vyhlásenie o prerušení

Pomocou príkazu `break` môžeme zastaviť slučku, aj keď je splnená podmienka while:

príklad

Opustite slučku, keď i je 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
1
2
```

```
3
```

Vyhlásenie o pokračovaní

Pomocou príkazu `pokračovať` môžeme zastaviť aktuálnu iteráciu a pokračovať ďalším:

príklad

Pokračujte na ďalšiu iteráciu, ak i je 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

```
1
2
4
5
6
```

Ostatné vyhlásenie

Pomocou príkazu `else` môžeme blok kódu spustiť raz, keď už nie je splnená podmienka:

príklad

Vytlačte správu, keď bude stav nepravdivý:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

```
1
2
```

```
3
4
5
i is no longer less than 6
```

Python pre slučky

`for` slučka sa používa pre iteráciu cez sekvencie (ktorá je buď zoznam n-tica, slovník, sadu, alebo reťazec).
Toto je menej ako `for` kľúčové slovo v iných programovacích jazykoch a funguje skôr ako metóda iterátora, ktorá sa nachádza v iných objektovo orientovaných programovacích jazykoch.
So slučkou `for` môžeme vykonať množinu príkazov, raz pre každú položku v zozname, tuple, set atď.

príklad

Vytlačte každé ovocie v zozname ovocia:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
apple
banana
cherry
```

`Pre` slučka nevyžaduje indexovacie premennú vopred nastaviť.

Opakovanie cez reťazec

Aj reťazce sú iterovateľné objekty, obsahujú postupnosť znakov:

príklad

Opakujte písmená v slove „banán“:

```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

Vyhlásenie o prerušení

Pomocou príkazu `break` môžeme zastaviť cyklus predtým, ako prešiel všetkými položkami:

príklad

Ak `x`je banán, opusťte slučku :

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

príklad

Ak `x`je banán, opusťte slučku , tentoraz je však prestávka pred tlačou:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

```
apple
```

Vyhlásenie o pokračovaní

S `pokračovať` vyhlásenie môžeme zastaviť aktuálne iterácii slučky, a pokračovať s ďalšie:

príklad

Netlačiť banán:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

```
apple
cherry
```

Funkcia rozsahu ()

Na opakované opakovanie množiny kódu môžeme použiť funkciu `range ()` ,

Funkcia `range ()` vracia postupnosť čísiel, ktorá začína od 0 v predvolenom nastavení a od prírastkov po 1 (predvolená hodnota) a končí zadaným číslom.

príklad

Použitie funkcie range ():

```
for x in range(6):
  print(x)
```

```
0
1
2
3
4
5
```

Pamätajte, že `rozsah (6)` nie je 0 až 6, ale 0 až 5.

Funkcia `range () je` predvolená na 0 ako počiatočná hodnota, je však možné určiť počiatočnú hodnotu pridaním parametra: `range (2, 6)` , čo znamená hodnoty od 2 do 6 (okrem 6):

príklad

Pomocou parametra start:

```
for x in range(2, 6):
  print(x)
```

```
2
3
4
5
```

Funkcia `range ()` predvolene zvyšuje prírastok o 1, je však možné určiť hodnotu prírastku pridaním tretieho parametra: `range (2, 30, 3 )` :

príklad

Zvýšte postupnosť o 3 (predvolená hodnota je 1):

```
for x in range(2, 30, 3):
  print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

Inak pre slučku

`else` Kľúčové slovo vo `for` slučke udáva blok kódu, ktoré majú byť vykonané, keď je slučka je hotový:

príklad

Vytlačte všetky čísla od 0 do 5 a po skončení slučky vytlačte správu:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

Vnorené slučky

Vnorená slučka je slučka vo vnútri slučky.
„Vnútorná slučka“ sa vykoná raz pre každú iteráciu „vonkajšej slučky“:

príklad

Vytlačte každé prídavné meno pre každé ovocie:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Úspešné vyhlásenie

`for` slučky nemôžu byť prázdne, ale ak z nejakého dôvodu máte `for` slučku bez obsahu, do `pass` príkazu vložte príkaz, aby ste sa vyhli chybe.

príklad

```
for x in [0, 1, 2]:
  pass
# s prázdnou pre slučku, ako je táto, by spôsobil chybu bez príkazu pass
```

Funkcie Pythonu

Funkcia je blok kódu, ktorý sa spustí, len keď sa volá.
Do funkcie môžete preniesť údaje, známe ako parametre.
Výsledkom môže funkcia vrátiť dáta.

Vytvorenie funkcie

V Pythone je funkcia definovaná pomocou kľúčového slova `def` :

príklad

```
def my_function():
  print("Hello from a function")
```

Volanie funkcie

Ak chcete zavolať funkciu, použite názov funkcie nasledovaný zátvorkami:

príklad

```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

argumenty

Informácie môžu byť prenášané do funkcií ako argumenty.
Argumenty sú uvedené za názvom funkcie, v zátvorkách. Môžete pridať ľubovoľný počet argumentov, stačí ich oddeliť čiarkou.

Nasledujúci príklad má funkciu s jedným argumentom (fname). Pri volaní funkcie odovzdáme krstné meno, ktoré sa používa na vytlačenie celého mena:

príklad

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Argumenty sa v dokumentáciách Pythonu často skracujú na *args* .

Parametre alebo argumenty?

Parameter a *argumenty* výrazov sa dajú použiť na to isté: informácie, ktoré sa prenášajú do funkcie.

Z hľadiska funkcie:
Parameter je premenná uvedená v zátvorkách v definícii funkcie.
Argument je hodnota, ktorá sa pri volaní vyšle funkcii.

Počet argumentov

V predvolenom nastavení musí byť funkcia volaná so správnym počtom argumentov. Znamená to, že ak vaša funkcia očakáva 2 argumenty, musíte volať funkciu s 2 argumentmi, nie viac a nie menej.

príklad

Táto funkcia očakáva 2 argumenty a získa 2 argumenty:

```
def my_function(fname, lname):
  print(fname + " " + lname)
```

```
my_function("Emil", "Refsnes")
```
```
Emil Refsnes
```

Ak sa pokúsite volať funkciu s 1 alebo 3 argumentmi, zobrazí sa chyba:

príklad

Táto funkcia očakáva 2 argumenty, ale získa iba 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```
```
TypeError: my_function() missing 1 required
positional argument: 'lname'
```

Arbitrary Arguments, * args

Ak neviete, koľko argumentov sa do vašej funkcie odovzdá, do `*` definície funkcie pridajte pred názov parametra. Týmto spôsobom sa funkcia dostane *tica* argumentov, a môže podľa toho pristupovať položky:

príklad

Ak počet argumentov nie je známy, *pred názov parametra pridajte znak:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Ľubovoľné argumenty sa v dokumentáciách Pythonu často skracujú na * *args* .

Argumenty kľúčových slov

Argumenty môžete poslať aj pomocou syntaxe *key* = *value* .

Týmto spôsobom nezáleží na poradí argumentov.

príklad

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3
= "Linus")
```

```
The youngest child is Linus
```

V Pythonovej dokumentácii sa veta *Argumenty kľúčových slov* často skracuje na *kwargs* .

Argumenty ľubovoľných kľúčových slov, ** kwargs

Ak neviete, koľko argumentov kľúčových slov sa do vašej funkcie odovzdá, pridajte `**` pred definíciu funkcie do definície funkcie dve hviezdičky:.
Týmto spôsobom funkcia získa *slovník* argumentov a podľa toho má prístup k položkám:

príklad

Ak počet argumentov kľúčových slov nie je známy, `**` pred názov parametra pridajte dvojitý znak:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

```
The youngest child is Linus
```

Arbitrary Kword Argumenty sú v Pythonových dokumentáciách často skracované na ** *kwargs* .

Predvolená hodnota parametra

Nasledujúci príklad ukazuje, ako použiť predvolenú hodnotu parametra.

Ak funkciu nazývame bez argumentu, použije predvolenú hodnotu:

príklad

```
def my_function(country = "Norway"):
  print("I am from " + country)
my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
His last name is Refsnes
```

Predloženie zoznamu ako argumentu

Na funkciu môžete poslať ľubovoľný dátový typ argumentu (reťazec, číslo, zoznam, slovník atď.) A vo funkcii sa bude považovať za rovnaký typ údajov.
Napríklad, ak pošlete zoznam ako argument, bude to stále zoznam, keď dosiahne funkciu:

príklad

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
apple
banana
cherry
```

Návratové hodnoty

Ak chcete, aby funkcia vrátila hodnotu, použite `return` príkaz:

príklad

```
def my_function(x):
  return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Spustiť príklad »

Úspešné vyhlásenie

`function` Definície nemôžu byť prázdne, ale ak z nejakého dôvodu máte `function` definíciu bez obsahu, vložte do `pass` príkazu príkaz, aby ste sa vyhli chybe.

príklad

```
def myfunction:
  pass
```

rekurzia

Python tiež akceptuje funkčnú rekurziu, čo znamená, že definovaná funkcia sa môže nazývať sama.
Rekurzia je bežný matematický a programový koncept. To znamená, že funkcia sa volá sama. To má tú výhodu, že môžete dosiahnuť opakovanie údajov. Vývojár by mal byť pri rekurzii veľmi opatrný, pretože je celkom ľahké preniknúť do funkcie, ktorá sa nikdy neskončí, alebo funkciu, ktorá využíva nadmerné množstvo pamäte alebo energie procesora. Avšak pri správnom písaní môže byť rekurzia veľmi efektívnym a matematicky elegantným prístupom k programovaniu. V tomto príklade
je `tri_recursion()` funkcia, ktorú sme sami definovali, aby sa nazývala sama („recurse“). Premennú `k` používame ako dáta, ktoré sa znižujú (`-1`) zakaždým, keď sa opakujeme. Rekurzia končí, keď stav nie je väčší ako 0 (tj keď je 0).

Novému vývojárovi môže trvať nejaký čas, kým zistí, ako presne to funguje. Najlepším spôsobom, ako to zistiť, je jeho testovanie a zmena.

príklad

Príklad rekurzie

```
def tri_recursion(k):
  if(k>0):
    result = k+tri_recursion(k-1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

```
Recursion Example Results
1
3
6
10
15
21
```

Python Lambda

Funkcia lambda je malá anonymná funkcia.
Funkcia lambda môže mať ľubovoľný počet argumentov, ale môže mať iba jeden výraz.

syntax

`lambda` *`arguments`* `:` *`expression`*

Vykoná sa výraz a vráti sa výsledok:

príklad

Funkcia lambda, ktorá pridáva 10 k číslu odovzdanému ako argument a tlačí výsledok:

```
x = lambda a : a + 10
print(x(5))
```

```
15
```

Lambda funkcie môžu mať ľubovoľný počet argumentov:

príklad

Funkcia lambda, ktorá násobí argument a argumentom ba a vytlačí výsledok:

```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

príklad

Funkcia lambda, ktorá sumarizuje argument a, b, ac a vytlačí výsledok:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
13
```

Prečo používať funkcie Lambda?

Sila lambda je lepšie zobrazená, keď ich používate ako anonymnú funkciu v rámci inej funkcie.
Povedzme, že máte definíciu funkcie, ktorá má jeden argument a tento argument sa vynásobí neznámym číslom:

```
def myfunc(n):
  return lambda a : a * n
```

Pomocou tejto definície funkcie vytvorte funkciu, ktorá vždy zdvojnásobí číslo, ktoré pošlete:

príklad

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
```

```
print(mydoubler(11))
```

```
22
```

Alebo pomocou rovnakej definície funkcie vytvorte funkciu, ktorá vždy *strojnásobí* číslo, ktoré pošlete:

príklad

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

```
33
```

Alebo použite rovnaké definície funkcií na vytvorenie oboch funkcií v rovnakom programe:

príklad

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

```
22
33
```

Použite funkcie lambda, ak je na krátky čas potrebná anonymná funkcia

Polia Python

Poznámka: Python nemá zabudovanú podporu pre polia, ale namiesto neho sa môžu používať zoznamy Python .

polia

Polia sa používajú na ukladanie viacerých hodnôt do jednej premennej:

príklad

Vytvorte pole obsahujúce názvy automobilov:

```
cars = ["Ford", "Volvo", "BMW"]
```

```
['Ford', 'Volvo', 'BMW']
```

Čo je to pole?

Pole je špeciálna premenná, ktorá dokáže uchovávať naraz viac ako jednu hodnotu.

Ak máte zoznam položiek (napríklad zoznam názvov automobilov), ukladanie automobilov do jednotlivých premenných by mohlo vyzerať takto:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

Čo ak však chcete prebehnúť cez autá a nájsť konkrétne vozidlo? A čo keby ste nemali 3 autá, ale 300?

Riešením je pole!

Pole môže obsahovať mnoho hodnôt pod jedným menom a hodnoty môžete získať prístupom k číslu indexu.

Prístup k prvkom poľa

Odkazujete na prvok poľa odkazom na *číslo indexu* .

príklad

Získajte hodnotu prvej položky poľa:

```
x = cars[0]
```

```
Ford
```

príklad

Upravte hodnotu prvej položky poľa:

```
cars[0] = "Toyota"
```

```
['Toyota', 'Volvo', 'BMW']
```

Dĺžka poľa

Použite `len()` metódu na vrátenie dĺžky poľa (počet prvkov v poli).

príklad

Vráťte počet prvkov v `cars` poli:

```
x = len(cars)
```

```
3
```

Poznámka: Dĺžka poľa je vždy o jeden viac ako najvyšší index poľa.

Slučkové prvky poľa

Pomocou `for in`slučky môžete prechádzať cez všetky prvky poľa.

príklad

Vytlačte každú položku v `cars` poli:

```
for x in cars:
  print(x)
```

```
Ford
Volvo
BMW
```

Pridávanie prvkov poľa

Túto `append()` metódu môžete použiť na pridanie prvku do poľa.

príklad

Pridajte do `cars` poľa ešte jeden prvok :

```
cars.append("Honda")
```

```
['Ford', 'Volvo', 'BMW', 'Honda']
```

Odstránenie prvkov poľa

Túto `pop()` metódu môžete použiť na odstránenie prvku z poľa.

príklad

Odstráňte druhý prvok `cars` poľa:

```
cars.pop(1)
```

```
['Ford', 'BMW']
```

Túto `remove()` metódu môžete použiť aj na odstránenie prvku z poľa.

príklad

Odstráňte prvok, ktorý má hodnotu „Volvo“:

```
cars.remove("Volvo")
```

```
['Ford', 'BMW']
```

Poznámka: Metóda zoznamu `remove()` odstráni iba prvý výskyt zadanej hodnoty.

Metódy poľa

Python má sadu vstavaných metód, ktoré môžete použiť v zoznamoch / poliach.

append() Pridá prvok na koniec zoznamu
clear() Odstráni všetky prvky zo zoznamu
copy() Vráti kópiu zoznamu
count() Vráti počet prvkov so zadanou hodnotou
extend() Na koniec aktuálneho zoznamu pridajte prvky zoznamu (alebo iné iterovateľné položky)
index() Vracia index prvého prvku so zadanou hodnotou
insert() Pridá prvok na určené miesto
pop() Odstráni prvok na určenej pozícii
remove() Odstráni prvú položku so zadanou hodnotou
reverse() Obráti poradie zoznamu
sort() Zoraďuje zoznam

Triedy a objekty Pythonu

Triedy / objekty Pythonu

Python je objektovo orientovaný programovací jazyk.
Takmer všetko v Pythone je objekt, jeho vlastnosti a metódy.
Trieda je ako konštruktor objektov alebo „plán" na vytváranie objektov.

Vytvorte triedu

Ak chcete vytvoriť triedu, použite kľúčové slovo `class`:

príklad

Vytvorte triedu s názvom MyClass s vlastnosťou s názvom x:

```
class MyClass:
  x = 5
```

```
<class '__main__.MyClass'>
```

Vytvorenie objektu

Teraz môžeme na vytváranie objektov použiť triedu s názvom myClass:

príklad

Vytvorte objekt s názvom p1 a vytlačte hodnotu x:

```
p1 = MyClass()
print(p1.x)
```

```
5
```

Funkcia __init __ ()

Vyššie uvedené príklady sú triedy a objekty v ich najjednoduchšej podobe a nie sú v praxi užitočné.

Aby sme pochopili význam tried, musíme pochopiť zabudovanú funkciu __init __ ().
Všetky triedy majú funkciu nazývanú __init __ (), ktorá sa vždy vykonáva pri iniciovaní triedy.
Pomocou funkcie __init __ () môžete priradiť hodnoty vlastnostiam objektu alebo iným operáciám, ktoré sú potrebné pri vytváraní objektu:

príklad

Vytvorte triedu s názvom Osoba, pomocou funkcie __init __ () priraďte hodnoty názvu a veku:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

```
John
36
```

Poznámka: `__init__()` funkcia sa nazýva automaticky zakaždým, keď sa trieda je použitá na vytvorenie nového objektu.

Metódy objektov

Objekty môžu obsahovať aj metódy. Metódy v objektoch sú funkcie, ktoré patria k objektu.
Vytvorme metódu v triede Osoba:

príklad

Vložte funkciu, ktorá vytlačí pozdrav a vykoná ho na objekte p1:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

Poznámka: `self` parameter je odkaz na aktuálnu inštanciu triedy, a používa sa pre prístup k premennej, ktoré patria do triedy.

Vlastný parameter

`self` Parameter je odkaz na aktuálny inštancie triedy, a používa sa pre prístup k premennej, ktoré patria do triedy. Nemusí byť pomenovaný `self`, môžete ho nazývať, čo sa vám páči, ale musí to byť prvý parameter akejkoľvek funkcie v triede:

príklad

Namiesto *seba* používajte slová *mysillyobject* a *abc* :

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

Upraviť vlastnosti objektu

Na objektoch, ako je tento, môžete upravovať vlastnosti:

príklad

Nastavte vek p1 na 40:

```
p1.age = 40
```

```
40
```

Odstrániť vlastnosti objektu

Vlastnosti objektov môžete odstrániť pomocou `del` kľúčového slova:

príklad

Odstráňte vlastnosť age z objektu p1:

```
del p1.age
```

```
AttributeError: 'Person' object has no attribute 'age'
```

Odstrániť objekty

Objekty môžete odstrániť pomocou `del` kľúčového slova:

príklad

Odstráňte objekt p1:

```
del p1
```

```
NameError: 'p1' is not defined
```

Úspešné vyhlásenie

`class` Definície nemôžu byť prázdne, ale ak z nejakého dôvodu máte `class` definíciu bez obsahu, vložte do `pass` príkazu príkaz, aby ste sa vyhli chybe.

príklad

```
class Person:
  pass
```

Dedičstvo Pythonu

Dedičnosť nám umožňuje definovať triedu, ktorá zdedí všetky metódy a vlastnosti z inej triedy.
Rodičovská trieda je trieda, z ktorej sa dedí, tiež sa nazýva základná trieda.
Podradená trieda je trieda, ktorá zdedila po inej triede, ktorá sa tiež nazýva odvodená trieda.

Vytvorte nadradenú triedu

Akákoľvek trieda môže byť nadradenou triedou, takže syntax je rovnaká ako pri vytváraní akejkoľvek inej triedy:

príklad

Vytvorte triedu s názvom `Person` s `firstname` a `lastname` vlastnosťami, a `printname` metódu:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then
execute the printname method:

x = Person("John", "Doe")
x.printname()
```

```
John Doe
```

Vytvorte triedu dieťaťa

Ak chcete vytvoriť triedu, ktorá zdedí funkčnosť z inej triedy, odošlite nadradenú triedu ako parameter pri vytváraní podradenej triedy:

príklad

Vytvorte triedu s názvom `Student`, ktorá zdedí vlastnosti a metódy od `Person` triedy:

```
class Student(Person):
  pass
```

Poznámka: `pass` Kľúčové slovo použite, ak do triedy nechcete pridať žiadne ďalšie vlastnosti alebo metódy.

Teraz má trieda Student rovnaké vlastnosti a metódy ako trieda Person.

príklad

Pomocou `Student` triedy vytvorte objekt a potom vykonajte `printname` metódu:

```
x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Pridajte funkciu __init __ ()

Doteraz sme vytvorili podradenú triedu, ktorá zdedí vlastnosti a metódy od svojho rodiča.

Chceme pridať `__init__()` funkciu do podradenej triedy (namiesto `pass` kľúčového slova).

Poznámka: `__init__()` funkcia sa nazýva automaticky zakaždým, keď sa trieda je použitá na vytvorenie nového objektu.

príklad

Pridajte `__init__()` funkciu do `Student` triedy:

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

Keď pridáte `__init__()` funkciu, podradená trieda už nebude zdediť funkciu rodiča `__init__()`.

Poznámka: dieťaťa `__init__()` funkcia **prepíše** dedičnosti rodičovskej `__init__()`funkciu.

Ak chcete zachovať dedičstvo funkcie rodiča `__init__()` , pridajte do funkcie rodiča hovor `__init__()`:

príklad

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

```
Mike Olsen
```

Teraz sme úspešne pridali funkciu __init __ () a ponechali sme dedičstvo nadradenej triedy a sme pripravení pridať funkciu do `__init__()`funkcie.

Použite funkciu super ()

Python má tiež `super()` funkciu, ktorá spôsobí, že podriadená trieda zdedí všetky metódy a vlastnosti od svojho rodiča:

príklad

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

```
Mike Olsen
```

Pri použití `super()` funkcie nemusíte používať názov rodičovského prvku, automaticky zdedí metódy a vlastnosti od svojho rodiča.

Pridať vlastnosti

príklad

Pridajte vlastnosť nazvanú `graduationyear` do `Student` triedy:

```
class Student(Person):
  def __init__(self, fname, lname):
```

```
    super().__init__(fname, lname)
    self.graduationyear = 2019
```

```
2019
```

V nasledujúcom príklade `2019` by mal byť rok premennou a mal by sa odovzdávať do `Student` triedy pri vytváraní študentských objektov. Ak to chcete urobiť, pridajte do funkcie __init __ () ďalší parameter:

príklad

Pridajte `year`parameter a pri vytváraní objektov zadajte správny rok:

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
```

```
2019
```

Pridajte metódy

príklad

Pridajte `welcome` do `Student` triedy metódu :

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)
```

```
Welcome Mike Olsen to the class of 2019
```

Ak do podradenej triedy pridáte metódu s rovnakým názvom ako funkcia v nadradenej triede, dedičstvo nadradenej metódy sa prepíše.

Iterátory Pythonu

Iterátor je objekt, ktorý obsahuje spočítateľný počet hodnôt.
Iterátor je objekt, ktorý je možné iterovať, čo znamená, že môžete prechádzať všetkými hodnotami.
Technicky je v Pythone iterátor objekt, ktorý implementuje protokol iterátora, ktorý pozostáva z metód `__iter__()` a `__next__()`.

Iterator vs Iterable

Zoznamy, n-tice, slovníky a množiny sú iterovateľné objekty. Sú to iterovateľné *kontajnery, z* ktorých môžete získať iterátor.
Všetky tieto objekty majú `iter()` metódu, ktorá sa používa na získanie iterátora:

príklad

Vráťte iterátor z n-tice a vytlačte každú hodnotu:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

```
apple
banana
cherry
```

Aj reťazce sú iterovateľné objekty a môžu vrátiť iterátor:

príklad

Reťazce sú tiež iterovateľné objekty, ktoré obsahujú postupnosť znakov:

```
mystr = "banana"
myit = iter(mystr)
```

```
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

```
b
a
n
a
n
a
```

Opakovanie cez Iterátor

Môžeme tiež použiť `for` slučku na iteráciu cez iterovateľný objekt:

príklad

Iterujte hodnoty n-tice:

```
mytuple = ("apple", "banana", "cherry")

for x in mytuple:
  print(x)
```

```
apple
banana
cherry
```

príklad

Iterujte znaky reťazca:

```
mystr = "banana"

for x in mystr:
  print(x)
```

```
b
a
n
a
n
a
```

`for` Slučka v skutočnosti vytvorí objekt iterátor a vykoná ďalšie metódy () pre každú slučku.

Vytvorte Iterátor

Ak chcete vytvoriť objekt / triedu ako iterátor, musíte implementovať metódy `__iter__()`a `__next__()` svoj objekt. Ako ste sa naučili v kapitole Python Classes / Objects , všetky triedy majú funkciu nazvanú `__init__()` , ktorá vám umožňuje vykonať nejaké inicializácie pri vytváraní objektu.

`__iter__()` Metóda sa správa podobne ako môžete vykonať operácie (inicializácia atď.), Ale musí byť vždy vrátiť objekt Iterator sám.

Táto `__next__()` metóda vám tiež umožňuje vykonávať operácie a musí vrátiť ďalšiu položku v poradí.

príklad

Vytvorte iterátor, ktorý vracia čísla, počnúc 1 a každá sekvencia sa zvýši o jednu (vráti sa 1,2,3,4,5 atď.):

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)
```

```
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

```
1
2
3
4
5
```

výnimku StopIteration

Vyššie uvedený príklad by pokračoval navždy, ak by ste mali dosť ďalších príkazov () alebo ak by sa použil v `for` slučke.
Aby sme zabránili opakovaniu iterácie navždy, môžeme použiť `StopIteration` vyhlásenie.
V `__next__()`metóde môžeme pridať podmienku ukončenia, ktorá vyvolá chybu, ak je iterácia vykonaná v špecifikovanom počte opakovaní:

príklad

Zastaviť po 20 iteráciách:

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration
```

```
myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Rozsah

Premenná je k dispozícii iba zvnútra oblasti, v ktorej je vytvorená. Toto sa nazýva **rozsah** .

Miestny rozsah

Premenná vytvorená vo funkcii patrí do *lokálneho rozsahu* tejto funkcie a môže sa použiť iba v rámci tejto funkcie.

príklad

Vo vnútri funkcie je k dispozícii premenná vytvorená vo funkcii:

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

```
300
```

Funkcia vo funkcii.

Ako je vysvetlené v príklade vyššie, premenná `x`nie je k dispozícii mimo funkcie, ale je k dispozícii pre akúkoľvek funkciu vo funkcii:

príklad

Lokálna premenná je prístupná z funkcie v rámci funkcie:

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

```
300
```

Globálny rozsah

Premenná vytvorená v hlavnej časti kódu Python je globálna premenná a patrí do globálneho rozsahu.
Globálne premenné sú dostupné z ľubovoľného rozsahu, globálnych a miestnych.

príklad

Premenná vytvorená mimo funkcie je globálna a môže ju použiť ktokoľvek:

```
x = 300

def myfunc():
```

```
  print(x)
myfunc()
print(x)
```
```
300
300
```

Premenné názvov

Ak pracujete s rovnakým názvom premennej vnútri a mimo funkcie, Python ich bude považovať za dve samostatné premenné, jednu dostupnú v globálnom rozsahu (mimo funkcie) a jednu dostupnú v lokálnom rozsahu (vo vnútri funkcie):

príklad

Táto funkcia vytlačí miestne `x`a potom kód vytlačí globálne `x`:

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```
```
200
300
```

Globálne kľúčové slovo

Ak potrebujete vytvoriť globálnu premennú, ale sú zaseknutí v miestnom rozsahu, môžete použiť `global` kľúčové slovo.
`global` Kľúčové slovo je premenná globálne.

príklad

Ak použijete `global` kľúčové slovo, premenná patrí do globálneho rozsahu:

```
def myfunc():
  global x
  x = 300
```

```
myfunc()

print(x)
```
```
300
```

`global` Kľúčové slovo tiež použite, ak chcete zmeniť globálnu premennú vo vnútri funkcie.

príklad

Ak chcete zmeniť hodnotu globálnej premennej vo funkcii, pozrite si premennú pomocou `global` kľúčového slova:

```
x = 300

def myfunc():
  global x
  x = 200

myfunc()
print(x)
```
```
200
```

Moduly Python

Čo je modul?

Zvážte modul ako rovnaký ako knižnica kódov.
Súbor obsahujúci sadu funkcií, ktoré chcete zahrnúť do svojej aplikácie.

Vytvorte modul

Ak chcete vytvoriť modul, jednoducho uložte požadovaný kód do súboru s príponou `.py`:

príklad

Tento kód uložte do súboru s názvom `mymodule.py`

```
def greeting(name):
  print("Hello, " + name)
```

Použite modul

Teraz môžeme použiť modul, ktorý sme práve vytvorili, pomocou `import` príkazu:

príklad

Importujte modul s názvom mymodule a zavolajte pozdravnú funkciu:

```
import mymodule
mymodule.greeting("Jonathan")
```

```
Hello, Jonathan
```

Poznámka: Ak používate funkciu z modulu, použite syntax: *názov_modulu.funkcie_funkcie* .

Premenné v module

Modul môže obsahovať funkcie, ako už boli opísané, ale aj premenné všetkých typov (polia, slovníky, objekty atď.):

príklad

Tento kód uložte do súboru `mymodule.py`

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

príklad

Importujte modul s názvom mymodule a získajte prístup k slovníku person1:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

```
36
```

Pomenovanie modulu

Súbor modulu môžete pomenovať ľubovoľne, ale musí mať príponu súboru `.py`

Premenovanie modulu

Pri importovaní modulu môžete vytvoriť alias pomocou `as`kľúčového slova:

príklad

Vytvorte alias pre `mymodule` volaný `mx`:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

```
36
```

Vstavané moduly

V Pythone je niekoľko vstavaných modulov, ktoré môžete kedykoľvek importovať.

príklad

Importovať a používať `platform` modul:

```
import platform

x = platform.system()
print(x)
```

```
Windows
```

Použitie funkcie dir ()

K dispozícii je vstavaná funkcia na výpis všetkých názvov funkcií (alebo názvov premenných) v module. `dir()`funkcia:

príklad

Zoznam všetkých definovaných názvov patriacich do modulu platformy:

```
import platform

x = dir(platform)
print(x)
```

```
['DEV_NULL', '_UNIXCONFDIR', 'WIN32_CLIENT_RELEASES',
'WIN32_SERVER_RELEASES', '__builtins__', '__cached__',
'__copyright__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', '__version__',
'_default_architecture', '_dist_try_harder',
'_follow_symlinks', '_ironpython26_sys_version_parser',
'_ironpython_sys_version_parser', '_java_getprop',
'_libc_search', '_linux_distribution',
'_lsb_release_version', '_mac_ver_xml', '_node',
'_norm_version', '_perse_release_file', '_platform',
'_platform_cache', '_pypy_sys_version_parser',
'_release_filename', '_release_version',
'_supported_dists', '_sys_version',
'_sys_version_cache', '_sys_version_parser',
'_syscmd_file', '_syscmd_uname', '_syscmd_ver',
'_uname_cache', '_ver_output', 'architecture',
'collections', 'dist', 'java_ver', 'libc_ver',
'linux_distribution', 'mac_ver', 'machine', 'node',
'os', 'platform', 'popen', 'processor', 'python_branch',
'python_build', 'python_compiler',
'python_implementation', 'python_revision',
'python_version', 'python_version_tuple', 're',
'release', 'subprocess', 'sys', 'system',
'system_aliases', 'uname', 'uname_result', 'version',
'warnings', 'win32_ver']
```

Poznámka: Funkcia dir () sa dá použiť na *všetky* moduly, aj tie, ktoré sami vytvoríte.

Importovať z modulu

Môžete sa rozhodnúť importovať iba časti z modulu pomocou `from` kľúčového slova.

príklad

Pomenovaný modul `mymodule` má jednu funkciu a jeden slovník:

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

príklad

Importovať iba modul person1 z modulu:

```
from mymodule import person1

print (person1["age"])
```

```
36
```

Poznámka: Pri importe pomocou `from` kľúčového slova nepoužívajte názov modulu, keď odkazujete na prvky v module. Príklad: `person1["age"]`, **nie** ~~`mymodule.person1["age"]`~~

Python Datetime

Dáta Pythona

Dátum v Pythone nie je jeho vlastný dátový typ, ale môžeme importovať modul pomenovaný `datetime` pre prácu s dátumami ako dátové objekty.

príklad
Importujte modul datetime a zobrazte aktuálny dátum:

```
import datetime

x = datetime.datetime.now()
print(x)
```

```
2020-01-02 18:22:58.406338
```

Dátum výstupu
Keď spustíme kód z vyššie uvedeného príkladu, výsledkom bude:

`2020-01-02 18:22:01.791924`

Dátum obsahuje rok, mesiac, deň, hodinu, minútu, sekundu a mikrosekundu.

`datetime` Modul má mnoho metód vrátiť informácie o objekte date.

Tu je niekoľko príkladov, o nich sa dozviete viac v tejto kapitole:

príklad
Vráťte rok a názov dňa v týždni:

```
import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

```
2020
Thursday
```

Vytváranie objektov dátumu
Na vytvorenie dátumu môžeme použiť `datetime()`triedu (konštruktor) `datetime`modulu.

`datetime()`Trieda vyžaduje tri parametre pre vytvorenie dátum: rok, mesiac, deň.

príklad

Vytvorte objekt dátumu:

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

```
2020-05-17 00:00:00
```

`datetime()` Trieda berie parametrov pre čas a časové pásmo (hodiny, minúty, sekundy, mikrosekundu, tzone), ale sú voliteľné, a má predvolenú hodnotu `0`, (`None` pre časové pásmo).

Metóda strftime ()

`datetime` Objekt má metódu pre formátovanie dáta objektov do čitateľných reťazca.

Metóda sa nazýva `strftime()` a má jeden parameter `format` na určenie formátu vráteného reťazca:

príklad

Zobraziť názov mesiaca:

```
import datetime
x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

```
June
```

Odkaz na všetky kódy zákonného formátu:

%a deň v týždni, krátka verzia *st*
%A Deň v týždni, plná verzia *streda*
% w Deň v týždni ako číslo 0-6, *0 je nedeľa 3*
% d deň v mesiaci *01-31 31*
% b Názov mesiaca, krátka verzia *Dec*
% B Názov mesiaca, plná verzia *December*
% m Mesiac ako číslo *01-12 12*
% y Rok, krátka verzia, bez storočia *18*

% Y Rok, úplná verzia *2018*
% H hodina *00-23 17*
% I hodina *00-12 05*
% P AM / *PM PM*
% M Minúta *00-59 41*
% S, druhý *00-59 08*
% f mikrosekundy *000000-999999 548513*
%z UTC *+0100*
%Z časová zona *CST*
% j Denné číslo roku *001-366 365*
% U Číslo týždňa v roku, nedeľa ako prvý deň v týždni, *00-53 52*
% W Číslo týždňa v roku, pondelok ako prvý deň v týždni, *00-53 52*
% c Miestna verzia dátumu a času po *31.12. 17:41:00 2018*
% x Miestna verzia dátumu *31/18/18*
% X Miestna verzia času *17:41:00*
%% a% character *%*

Python RegEx

RegEx alebo regulárny výraz je postupnosť znakov, ktoré tvoria vyhľadávací vzor.
RegEx možno použiť na kontrolu, či reťazec obsahuje zadaný vyhľadávací vzor.

Modul RegEx

Python má zabudovaný balík nazvaný `re`, ktorý je možné použiť na prácu s regulárnymi výrazmi.
Import `re` modulu:

```
import re
```

RegEx v Pythone

Po importovaní `re`modulu môžete začať používať regulárne výrazy:

príklad

Vyhľadajte reťazec, aby ste zistili, či začína reťazcom „The“ a končí reťazcom „Spain“:

```
import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

Funkcie RegEx

`re` Modul ponúka rad funkcií, ktoré nám umožňuje vyhľadávať reťazec na zápas:

indall Vráti zoznam obsahujúci všetky zhody
search Vráti objekt Match, ak sa v reťazci nachádza zhoda
split Vráti zoznam, v ktorom bol reťazec rozdelený pri každej zhode
sub Nahradí jednu alebo viac zhôd reťazcom

metaznaky

Metaznaky sú znaky so špeciálnym významom:

[] Sada znakov „[a-m]“

**** Signalizuje špeciálnu sekvenciu (môže sa použiť aj na únik zo špeciálnych znakov) "\ d"

, Ľubovoľný znak (okrem znaku nového riadku) „he..o“

^ Začína sa „^ ahoj“

$ Končí „world $“

***** Nula alebo viac výskytov „aix *“

+ Jeden alebo viac výskytov „aix +“

{} Presne určený počet výskytov "al {2}"

| Buď alebo „spadne“ zostane

() Zachytenie a skupina

Špeciálne sekvencie

Po špeciálnej postupnosti \nasleduje jeden zo znakov v zozname nižšie a má osobitný význam:

\ A Vráti zhodu, ak sú zadané znaky na začiatku reťazca "\ AThe"
\ b Vráti zhodu, kde sú zadané znaky na začiatku alebo na konci slova r "\ bain"
r "ain \ b"
\ B Vráti zhodu, kde sú zadané znaky, ale NIE na začiatku (alebo na konci) slova r "\ Bain"
r "ain \ B"
\ d Vráti zhodu, kde reťazec obsahuje číslice (čísla od 0 do 9) "\ d"
\ D Vráti zhodu, keď reťazec NEBUDE obsahovať číslice "\ D"
\ s Vracia zhodu, keď reťazec obsahuje znak medzery "\ s"
\ S Vráti zhodu, keď reťazec NEBOLA znak medzery "\ S"
\ w Vráti zhodu, v ktorej reťazec obsahuje akékoľvek slovo (znaky od a do Z, číslice od 0 do 9 a znak podčiarknutia _) "\ w"
\ W Vráti zhodu, v ktorej reťazec NEBUDE obsahovať žiadne znaky slov "\ W"
\ Z Vráti zhodu, ak sú zadané znaky na konci reťazca "Španielsko \ Z"

súpravy

Sada je sada znakov vo vnútri hranatých zátvoriek []so špeciálnym významom:

[arn] Vracia zhodu, ak je prítomný jeden zo zadaných znakov (a, r alebo n)

[a-n] Vracia zhodu pre malé písmená abecedne medzi a a n

[^ arn] Vracia zhodu ľubovoľného znaku s VÝNIMKOU a, r a n

Vracia zhodu, ak je prítomná ktorákoľvek zo špecifikovaných číslic (0, 1, 2 alebo 3)

[0-9] Vracia zhodu ktorejkoľvek číslice medzi 0 a 9

[0-5] [0-9] Vracia zhodu pre ľubovoľné dvojciferné čísla od 00 do 59

[a-zA-Z] Vracia zhodu ľubovoľného znaku podľa abecedy medzi a a z, malými písmenami alebo veľkými písmenami

[+] V množinách +, *,., |, (), $, {} Nemá žiadny zvláštny význam, takže [+] znamená: vráti zhodu pre ľubovoľný znak + v reťazci

Funkcia findall ()

`findall()` Funkcia vracia zoznam obsahujúci všetky zápasy.

príklad

Vytlačte zoznam všetkých zápasov:

```
import re

str = "The rain in Spain"
x = re.findall("ai", str)
print(x)
```

```
['ai', 'ai']
```

Zoznam obsahuje zhody v poradí, v akom boli nájdené.
Ak sa nenájdu žiadne zhody, vráti sa prázdny zoznam:

príklad

Ak sa nenájde žiadna zhoda, vráťte prázdny zoznam:

```
import re

str = "The rain in Spain"
```

```
x = re.findall("Portugal", str)
print(x)
```

```
[]
No match
```

Funkcia vyhľadávania ()

`search()` Funkcia vyhľadá reťazec, či obsahuje zhodu a vracia objekt zápasu , ak existuje zhoda.
Ak existuje viac ako jeden zápas, bude vrátený iba prvý výskyt zápasu:

príklad

Vyhľadajte prvý znak medzery v reťazci:

```
import re

str = "The rain in Spain"
x = re.search("\s", str)

print("The first white-space character is located in position:", x.start())
```

```
The first white-space character is located in position:
3
```

Ak sa nenájdu žiadne zhody, `None`vráti sa hodnota :

príklad

Vykonajte vyhľadávanie, ktoré nevracia žiadnu zhodu:

```
import re

str = "The rain in Spain"
x = re.search("Portugal", str)
print(x)
```

```
None
```

Funkcia split ()

`split()`Funkcia vracia zoznam kde reťazec bola rozdelená na každý zápas:

príklad

Rozdeľte pri každom znaku medzery:

```
import re

str = "The rain in Spain"
x = re.split("\s", str)
print(x)
```

```
['The', 'rain', 'in', 'Spain']
```

Počet výskytov môžete ovládať zadaním `maxsplit` parametra:

príklad

Rozdeľte reťazec iba pri prvom výskyte:

```
import re

str = "The rain in Spain"
x = re.split("\s", str, 1)
print(x)
```

```
['The', 'rain in Spain']
```

Funkcia sub ()

`sub()` Funkcii nahradí zápasu s textom podľa Vášho výberu:

príklad

Nahraďte každý znak medzery číslom 9:

```
import re

str = "The rain in Spain"
x = re.sub("\s", "9", str)
print(x)
```

```
The9rain9in9Spain
```

Počet nahradení môžete ovládať zadaním `count` parametra:

príklad

Vymeňte prvé 2 výskyty:

```
import re

str = "The rain in Spain"
x = re.sub("\s", "9", str, 2)
print(x)
```

```
The9rain9in Spain
```

Objekt zápasu

Match objekt je objekt obsahujúci informácie o vyhľadávaní a výsledku.

Poznámka: Ak neexistuje zhoda, hodnota `None`sa vráti namiesto objektu Zhoda.

príklad

Vykonajte vyhľadávanie, ktoré vráti objekt zápasu:

```
import re

str = "The rain in Spain"
x = re.search("ai", str)
print(x) #this will print an object
```

```
<_sre.SRE_Match object; span=(5, 7), match='ai'>
```

Objekt Zhoda má vlastnosti a metódy použité na získanie informácií o vyhľadávaní a výsledku:

`.span()` vráti n-ticu obsahujúcu začiatočnú a koncovú polohu zápasu.

`.string` vráti reťazec odovzdaný do funkcie

`.group()` vráti časť reťazca, kde bola zhoda

príklad

Vytlačte polohu (počiatočnú a koncovú polohu) prvého výskytu zápasu.

Regulárny výraz vyhľadáva slová, ktoré začínajú veľkými písmenami „S“:

```
import re
```

```
str = "The rain in Spain"
x = re.search(r"\bS\w+", str)
print(x.span())
```

```
(12, 17)
```

príklad

Vytlačte reťazec odovzdaný do funkcie:

```
import re

str = "The rain in Spain"
x = re.search(r"\bS\w+", str)
print(x.string)
```

```
The rain in Spain
```

príklad

Vytlačte časť reťazca, kde bola zhoda.
Regulárny výraz vyhľadáva slová, ktoré začínajú veľkými písmenami „S“:

```
import re

str = "The rain in Spain"
x = re.search(r"\bS\w+", str)
print(x.group())
```

```
Spain
```

Poznámka: Ak neexistuje zhoda, hodnota `None`sa vráti namiesto objektu Zhoda.

Python PIP

Čo je to PIP?

PIP je správca balíkov pre balíčky Python alebo moduly, ak chcete.

Poznámka: Ak máte Python verzie 3.4 alebo novšej, štandardne je zahrnutý PIP.

Čo je to balík?

Balík obsahuje všetky súbory, ktoré potrebujete pre modul. Moduly sú knižnice kódov Python, ktoré môžete zahrnúť do svojho projektu.

Skontrolujte, či je nainštalovaný PIP

Prejdite z príkazového riadka do umiestnenia adresára skriptov Pythonu a zadajte nasledujúci text:

príklad

Skontrolujte verziu PIP:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version
```

Nainštalujte program PIP

Ak nemáte nainštalovaný PIP, môžete si ho stiahnuť a nainštalovať z tejto stránky: https://pypi.org/project/pip/

Stiahnite si balík

Stiahnutie balíka je veľmi jednoduché.
Otvorte rozhranie príkazového riadka a povedzte PIP, aby stiahol požadovaný balík.
Prejdite z príkazového riadka do umiestnenia adresára skriptov Pythonu a zadajte nasledujúci text:

príklad

Stiahnite si balík s názvom "camelcase":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase
```

Teraz ste si stiahli a nainštalovali svoj prvý balík!

Používanie balíka

Po nainštalovaní balíka je balík pripravený na použitie. Naimportujte balík „camelcase“ do svojho projektu.

príklad

Importovať a používať výraz „ťava“:

```
import camelcase

c = camelcase.CamelCase()

txt = "hello world"

print(c.hump(txt))
```

```
Lorem Ipsum Dolor Sit Amet
```

Vyhľadajte balíky

Viac balíkov nájdete na https://pypi.org/ .

Vyberte balík

Pomocou `uninstall` príkazu odstráňte balík:

príklad

Odinštalujte balík s názvom "camelcase":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip uninstall camelcase
```

Správca balíkov PIP vás požiada, aby ste potvrdili, že chcete odstrániť balík ťavy:

```
Uninstalling camelcase-02.1:
  Would remove:
    c:\users\Your
Name\appdata\local\programs\python\python36-32\lib\site-
packages\camecase-0.2-py3.6.egg-info
    c:\users\Your
Name\appdata\local\programs\python\python36-32\lib\site-
packages\camecase\*
Proceed (y/n)?
```

Stlačte `y`a balík bude odstránený.

Zoznam balíkov

Pomocou `list`príkazu môžete zobraziť zoznam všetkých balíkov nainštalovaných vo vašom systéme:

príklad

Zoznam nainštalovaných balíkov:

```
C:\Users\Your
Name\AppData\Local\Programs\Python\Python36-
32\Scripts>pip list
```

result:

```
Package         Version
-----------------------
camelcase       0.2
mysql-connector 2.1.6
pip             18.1
pymongo         3.6.1
setuptools      39.0.1
```

Vyskúšajte Python, s výnimkou

`try` Blok umožňuje testovať blok kódu chyby.
`except` Blok umožňuje spracovať došlo k chybe.
`finally` Blok umožňuje spustenie kódu, bez ohľadu na výsledok z try- a okrem blokov.

Spracovanie výnimiek

Ak sa vyskytne chyba alebo výnimka, ako ju nazývame, Python sa zvyčajne zastaví a vygeneruje chybové hlásenie.
Tieto výnimky možno spracovať pomocou `try`príkazu:

príklad

`try` Blok bude generovať výnimku, pretože `x`nie je definovaný:

```
try:
  print(x)
except:
  print("An exception occurred")
```

```
An exception occurred
```

Pretože blok try vyvolá chybu, vykoná sa blok okrem.
Bez bloku vyskúšania sa program zrúti a vyvolá chybu:

príklad

Toto vyhlásenie vyvolá chybu, pretože `x`nie je definované:

```
print(x)
```

```
NameError: name 'x' is not defined
```

Mnoho výnimiek

Môžete definovať ľubovoľný počet blokov výnimiek, napr. Ak chcete vykonať špeciálny blok kódu pre špeciálny druh chyby:

príklad

Vytlačte jednu správu, ak blok vyskúšania vyvoláva a `NameError` a ďalšie pre ďalšie chyby:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

```
Variable x is not defined
```

inak

`else`Kľúčové slovo môžete použiť na definovanie bloku kódu, ktorý sa má vykonať, ak nedošlo k žiadnym chybám:

príklad

V tomto príklade `try`blok negeneruje žiadnu chybu:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

```
Hello
Nothing went wrong
```

konečne

`finally` Blok, ak je uvedené, bude vykonané bez ohľadu na to v prípade, že blok try vyvolá chybu, alebo nie.

príklad

```
try:
  print(x)
except:
```

```
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

```
Something went wrong
The 'try except' is finished
```

To môže byť užitočné pri zatváraní objektov a čistení zdrojov:

príklad

Pokúste sa otvoriť a zapísať do súboru, ktorý nemožno zapísať:

```
try:
  f = open("demofile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

```
Something went wrong when writing to the file
```

Program môže pokračovať bez toho, aby bol objekt súboru otvorený.

Zvýšte výnimku

Ako vývojár Pythonu si môžete zvoliť výnimku, ak sa vyskytne podmienka.

Na vyvolanie (alebo zvýšenie) výnimky použite `raise` kľúčové slovo.

príklad

Zvýšte chybu a zastavte program, ak je x menšie ako 0:

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

```
Traceback (most recent call last):
  File "demo_ref_keyword_raise.py", line 4, in <module>
    raise Exception("Sorry, no numbers below zero")
Exception: Sorry, no numbers below zero
```

`raise` Kľúčové slovo sa používa na zvýšenie výnimku.
Môžete definovať, aký druh chyby sa má zvýšiť a text, ktorý sa má používateľovi vytlačiť.

príklad

Ak x nie je celé číslo, zdvihnite TypeError:

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

```
Traceback (most recent call last):
  File "demo_ref_keyword_raise2.py", line 4, in <module>
    raise TypeError("Only integers are allowed")
TypeError: Only integers are allowed
```

Vstup používateľa Pythonu

Python umožňuje vstup od užívateľa.
To znamená, že sme schopní požiadať používateľa o vstup.
Táto metóda je v Pythone 3.6 trochu odlišná ako v Pythone 2.7.
Python 3.6 používa túto `input()` metódu.
Python 2.7 používa túto `raw_input()` metódu.
Nasledujúci príklad žiada o užívateľské meno a po zadaní užívateľského mena sa vytlačí na obrazovku:

Python 3.6

```
username = input("Enter username:")
print("Username is: " + username)
```

```
Enter username:
```

Python 2.7

```
username = raw_input("Enter username:")
print("Username is: " + username)
```

```
Enter username:
```

Python prestane vykonávať svoju činnosť, pokiaľ ide o `input()`funkciu, a pokračuje, keď užívateľ zadal nejaký vstup.

Formátovanie reťazcov Python

Aby sme sa ubezpečili, že sa reťazec bude zobrazovať podľa očakávania, môžeme výsledok formátovať pomocou tejto `format()` metódy.

Formát reťazca ()

Táto `format()` metóda umožňuje formátovať vybrané časti reťazca.
Niekedy sú časti textu, ktoré nekontrolujete, možno pochádzajú z databázy alebo zo vstupu používateľa?
Ak chcete tieto hodnoty ovládať, `{}` do textu pridajte zástupné symboly (zložené zátvorky) a pomocou `format()` metódy vykonajte tieto hodnoty :

príklad

Ak chcete zobraziť cenu, pridajte zástupný symbol:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

```
The price is 49 dollars
```

Do zložených zátvoriek môžete pridať parametre na určenie spôsobu prevodu hodnoty:

príklad

Naformátujte cenu, ktorá sa má zobraziť, ako číslo s dvoma desatinnými miestami:

```
txt = "The price is {:.2f} dollars"
```

```
The price is 49.00 dollars
```

Pozrite si všetky typy formátovania v našom referenčnom formáte String () .

Viaceré hodnoty

Ak chcete použiť viac hodnôt, pridajte do metódy format () ďalšie hodnoty:

```
print(txt.format(price, itemno, count))
```

A pridajte ďalšie zástupné symboly:

príklad

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item number 567 for 49.00 dollars.
```

Indexové čísla

Môžete použiť indexové čísla (číslo v zátvorkách `{0}`), aby ste sa uistili, že hodnoty sú umiestnené v správnych zástupných znakoch:

príklad

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item number 567 for 49.00 dollars.
```

Ak sa chcete na tú istú hodnotu odvolávať viackrát, použite indexové číslo:

príklad

```
age = 36
name = "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
```

```
His name is John. John is 36 years old.
```

Pomenované indexy

Pomenované indexy môžete použiť aj zadaním názvu do zložených zátvoriek `{carname}`, ale pri odovzdávaní hodnôt parametrov musíte použiť názvy `txt.format(carname = "Ford")`:

príklad

```
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname = "Ford", model
= "Mustang"))
```

```
I have a Ford, it is a Mustang.
```

Metóda formátu Python String ()

príklad

Vložte cenu do zástupného symbolu, cena by mala byť v pevnom bode, vo formáte dvoch desatinných miest:

```
txt = "For only {price:.2f} dollars!"
print(txt.format(price = 49))
```

Spustiť príklad »

Definícia a použitie

`format()`Metóda formátuje stanovenú hodnotu (y) a vložiť ich do vyhradeného miesta Reťazec je.

Zástupný symbol je definovaný pomocou zložených zátvoriek: {}. Viac informácií o zástupných symboloch nájdete v časti Zástupný symbol.
`format()` Metóda vracia formátovaný reťazec.

syntax

```
string.format(value1, value2...)
```

Hodnoty parametrov

value1, value2 ... Povinné. Jedna alebo viac hodnôt, ktoré by sa mali naformátovať a vložiť do reťazca. Hodnoty môžu byť číslo udávajúce pozíciu prvku, ktorý chcete odstrániť.

Hodnoty sú buď zoznam hodnôt oddelených čiarkami, zoznam kľúč = hodnota alebo ich kombinácia.

Hodnoty môžu byť ľubovoľného typu údajov.

Zástupcovia

Zástupné symboly možno identifikovať pomocou pomenovaných indexov `{price}`, očíslovaných indexov `{0}` alebo dokonca prázdnych zástupných symbolov `{}`.

príklad

Použitie rôznych zástupných hodnôt:

```
txt1 = "My name is {fname}, I'am {age}".format(fname = "John", age = 36)
txt2 = "My name is {0}, I'am {1}".format("John",36)
txt3 = "My name is {}, I'am {}".format("John",36)
```

```
My name is John, I'm 36
My name is John, I'm 36
My name is John, I'm 36
```

Typy formátovania

Vo vnútri zástupných symbolov môžete pridať formátovanie na formátovanie výsledku:

:< Doľava zarovná výsledok (v rámci dostupného priestoru)
:> Vpravo zarovnáva výsledok (v rámci dostupného priestoru)
: ^ Centrum zarovná výsledok (v rámci dostupného priestoru)
: = Znak umiestni doľava na najvyššiu pozíciu
: + Použite znamienko plus na označenie, či je výsledok pozitívny alebo negatívny
: - Znak mínus používajte iba pre záporné hodnoty
: Použite medzeru na vloženie medzery pred kladné čísla (a znamienko mínus pred záporné čísla)
:, Použite čiarku ako oddeľovač tisícok
: _ Ako oddeľovač tisícov použite podčiarkovník
: b Binárny formát
: c Skonvertuje hodnotu na zodpovedajúci znak Unicode
: d Desiatkový formát
: e Vedecký formát, s malým písmenom e
: E Vedecký formát, s veľkými písmenami E
: f Formát čísla bodu opravy
: F Formát čísla fixného bodu vo formáte veľkých písmen (inf a nan sa zobrazujú ako INF a NAN)
: g Všeobecný formát
: G Všeobecný formát (použitie veľkých písmen E pre vedecké zápisy)
: o osmičkový formát
: x hexadecimálny formát, malé písmená

: X Hex formát, veľké písmená

: n Formát čísla

:% Percentuálneho formátu

Python File Open

Správa súborov je dôležitou súčasťou každej webovej aplikácie.
Python má niekoľko funkcií na vytváranie, čítanie, aktualizáciu a mazanie súborov.

Spracovanie súborov

Kľúčovou funkciou pre prácu so súbormi v Pythone je `open()` funkcia.
`open()` Funkcia má dva parametre; *názov súboru* a *režim* .
Existujú štyri rôzne metódy (režimy) na otvorenie súboru:
`"r"` - Čítať - predvolená hodnota. Otvorí súbor na čítanie, chyba, ak súbor neexistuje
`"a"` - Pridať - otvorí súbor na pripojenie, vytvorí súbor, ak neexistuje
`"w"` - Write - otvorí súbor na zápis, vytvorí súbor, ak neexistuje
`"x"` - Vytvoriť - Vytvorí určený súbor a vráti chybu, ak existuje
Okrem toho môžete určiť, či sa má súbor spracovať ako binárny alebo textový režim
`"t"` - Text - predvolená hodnota. Textový režim
`"b"` - Binárny - Binárny režim (napr. Obrázky)

syntax

Na otvorenie súboru na čítanie stačí zadať názov súboru:

```
f = open("demofile.txt")
```

Vyššie uvedený kód je rovnaký ako:

```
f = open("demofile.txt", "rt")
```

Pretože `"r"` pre čítanie a `"t"` pre text sú predvolené hodnoty, nemusíte ich špecifikovať.

Poznámka: Skontrolujte, či súbor existuje, inak sa zobrazí chyba.

Python File Open

Otvorte súbor na serveri

Predpokladajme, že máme nasledujúci súbor, ktorý sa nachádza v rovnakom priečinku ako Python:

demofile.txt

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Na otvorenie súboru použite vstavanú `open()` funkciu.

`open()` Funkcia vracia objekt súbor, ktorý má `read()` metódu pre čítanie obsahu súboru:

príklad

```
f = open("demofile.txt", "r")
print(f.read())
```

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Čítať iba časti súboru

V predvolenom nastavení `read()` metóda vracia celý text, ale môžete tiež určiť, koľko znakov chcete vrátiť:

príklad

Vráťte 5 prvých znakov súboru:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

```
Hello
```

Prečítajte si riadky

Jeden riadok môžete vrátiť pomocou `readline()` metódy:

príklad

Prečítajte si jeden riadok súboru:

```
f = open("demofile.txt", "r")
print(f.readline())
```

```
Hello! Welcome to demofile.txt
```

Dvojitým volaním `readline()`si môžete prečítať prvé dva riadky:

príklad

Prečítajte si dva riadky súboru:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
```

Opakovaním v riadkoch súboru si môžete prečítať celý súbor, riadok po riadku:

príklad

Slučka cez súbor riadok po riadku:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck
```

Zatvorte súbory

Keď ste s tým hotoví, je dobré vždy tento súbor zavrieť.

príklad

Po dokončení súboru súbor zatvorte:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

```
Hello! Welcome to demofile.txt
```

Poznámka: Svoje súbory by ste mali vždy zavrieť, v niektorých prípadoch sa zmeny vykonané v súbore nemusia zobraziť, kým ho nezatvoríte.

Zápis súboru Python

Zápis do existujúceho súboru

Ak chcete zapísať do existujúceho súboru, musíte do `open()` funkcie pridať parameter :

`"a"` - Pripojiť - pripojí sa na koniec súboru

`"w"` - Write - prepíše akýkoľvek existujúci obsah

príklad

Otvorte súbor "demofile2.txt" a pripojte k nemu obsah:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

```
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

príklad

Otvorte súbor "demofile3.txt" a prepíšte obsah:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
```

```
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

```
Woops! I have deleted the content!
```

Poznámka: Metóda „w“ prepíše celý súbor.

Vytvorte nový súbor

Na vytvorenie nového súboru v Pythone použite `open()` metódu s jedným z nasledujúcich parametrov:

`"x"` - Vytvoriť - vytvorí súbor a vráti chybu, ak existuje
`"a"` - Pridať - vytvorí súbor, ak zadaný súbor neexistuje
`"w"` - Write - vytvorí súbor, ak zadaný súbor neexistuje

príklad

Vytvorte súbor s názvom „myfile.txt“:

```
f = open("myfile.txt", "x")
```

Výsledok: vytvorí sa nový prázdny súbor!

príklad

Vytvorte nový súbor, ak neexistuje:

```
f = open("myfile.txt", "w")
```

Vymazať súbor Python

Odstrániť súbor

Ak chcete odstrániť súbor, musíte importovať modul OS a spustiť jeho `os.remove()` funkciu:

príklad

Odstráňte súbor "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Skontrolujte, či existuje súbor:

Aby ste sa vyhli chybe, mali by ste si pred pokusom o jej odstránenie skontrolovať, či súbor existuje:

príklad

Skontrolujte, či súbor existuje, a *potom* ho odstráňte:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Odstrániť priečinok

Ak chcete odstrániť celý priečinok, použite tento `os.rmdir()`postup:

príklad

Odstráňte priečinok „myfolder“:

```
import os
os.rmdir("myfolder")
```

Poznámka: Môžete odstrániť iba *prázdne* priečinky.

Python Zabudované funkcie

Python má sadu vstavaných funkcií.

abs () Vráti absolútnu hodnotu čísla
all () Vráti true, ak sú všetky položky v iterovateľnom objekte pravdivé
any () Vráti true, ak je niektorá položka v iterovateľnom objekte pravdivá
ascii () Vracia čitateľnú verziu objektu. Nahradí znaky no-ascii znakom escape
bin () Vracia binárnu verziu čísla
bool () Vracia boolovskú hodnotu zadaného objektu
bytearray () Vracia pole bajtov
bytes () Vracia bajtový objekt
callable () Vráti hodnotu True, ak je určený objekt na volanie, inak False
chr () Vracia znak zo zadaného kódu Unicode.
classmethod () Konvertuje metódu na metódu triedy
compile () Vracia zadaný zdroj ako objekt pripravený na vykonanie
complex () Vracia komplexné číslo

delattr () Odstráni zadaný atribút (vlastnosť alebo metóda) zo zadaného objektu
dict () Vracia slovník (Array)
dir () Vráti zoznam vlastností a metód zadaného objektu
divmod () Vráti kvocient a zvyšok, keď je argument1 vydelený argumentom2
enumerate () Vezme kolekciu (napr. n-tica) a vráti ju ako enumerovaný objekt
eval () Vyhodnotí a vykoná výraz
exec () Vykoná zadaný kód (alebo objekt)
filter () Použite funkciu filtra na vylúčenie položiek v iterovateľnom objekte
float () Vráti číslo s pohyblivou rádovou čiarkou
format () Naformátuje zadanú hodnotu
frozenset () Vracia objekt frozenset
getattr () Vracia hodnotu zadaného atribútu (vlastnosť alebo metóda)
globals () Vráti aktuálnu tabuľku globálnych symbolov ako slovník
hasattr () Vráti hodnotu true, ak má určený objekt určený atribút (vlastnosť / metóda)
hash () Vracia hashovaciu hodnotu zadaného objektu
help () Spustí zabudovaný systém pomoci
hex () Skonvertuje číslo na hexadecimálnu hodnotu
id () Vracia id objektu
input () Povolenie vstupu používateľa
int () Vracia celé číslo

isinstance () Vráti true, ak je určený objekt inštanciou určitého objektu
issubclass () Vracia true, ak je zadaná trieda podtriedou zadaného objektu
iter () Vracia objekt iterátora
len () Vráti dĺžku objektu
list () Vracia zoznam
locals () Vráti aktualizovaný slovník aktuálnej tabuľky miestnych symbolov
map () Vracia zadaný iterátor so zadanou funkciou použitou pre každú položku
max () Vráti najväčšiu položku v iterovateľnom objekte
memoryview () Vracia objekt zobrazenia pamäte
min () Vráti najmenšiu položku z iterovateľnej položky
next () Vráti ďalšiu položku v iterovateľnom poradí
object () Vracia nový objekt
oct () Skonvertuje číslo na osmičkovú
open () Otvorí súbor a vráti objekt súboru
ord () Skonvertuje celé číslo predstavujúce Unicode zadaného znaku
pow () Vracia hodnotu x do sily y
print () Tlač na štandardné výstupné zariadenie
property () Získa, nastaví, vymaže vlastnosť
range () Vráti postupnosť čísiel, počínajúc 0 a prírastky 1 (predvolené nastavenie)
repr () Vracia čitateľnú verziu objektu
reversed() Vráti spätný iterátor

round () Zaokrúhli čísla

set () Vracia nový set objekt

setattr () Nastavuje atribút (vlastnosť / metóda) objektu

slice () Vracia objekt rezu

sort () Vracia usporiadaný zoznam

@staticmethod () Skonvertuje metódu na statickú metódu

str () Vracia reťazcový objekt

sum () Súčty položiek iterátora

super () Vracia objekt, ktorý predstavuje nadradenú triedu

tuple () Vracia n-ticu

type () Vracia typ objektu

vars () Vracia vlastnosť objektu __dict__

zip () Vracia iterátor z dvoch alebo viacerých iterátorov

Metódy reťazca Python

Python má sadu vstavaných metód, ktoré môžete použiť na reťazcoch.

Poznámka: Všetky metódy reťazcov vracajú nové hodnoty. Nemenia pôvodný reťazec.

capitalize () Skonvertuje prvý znak na veľké písmená

casefold () Skonvertuje reťazec na malé písmená

center () Vracia centrovaný reťazec

count () Vráti, koľkokrát sa zadaná hodnota vyskytne v reťazci
encode () Vráti kódovanú verziu reťazca
endwith () Vráti true, ak reťazec končí zadanou hodnotou
expandtabs () Nastavuje veľkosť záložky reťazca
find () Vyhľadá reťazec pre zadanú hodnotu a vráti polohu, kde sa našiel
format () Naformátuje zadané hodnoty v reťazci
format_map () Naformátuje zadané hodnoty v reťazci
index () Vyhľadá reťazec pre zadanú hodnotu a vráti polohu, kde sa našiel
isalnum () Vráti hodnotu true, ak sú všetky znaky v reťazci alfanumerické
isalpha () Vráti true, ak sú všetky znaky v reťazci v abecede
isdecimal () Vráti true, ak sú všetky znaky v reťazci desatinné miesta
isdigit () Vráti true, ak sú všetky znaky v reťazci číslice
isidentifier () Vráti true, ak je reťazec identifikátor
islower () Vracia true, ak sú všetky znaky v reťazci malé
isnumeric () Vráti hodnotu true, ak sú všetky znaky v reťazci číselné
isprintable () Vráti true, ak je možné tlačiť všetky znaky v reťazci
isspace () Vráti true, ak všetky znaky v reťazci sú medzery
istitle () Vráti true, ak sa reťazec riadi pravidlami názvu
isupper () Vracia true, ak sú všetky znaky v reťazci veľké

join () Pripojí prvky iterovateľné na koniec reťazca
ljust () Vracia ľavú odôvodnenú verziu reťazca
Lower () Skonvertuje reťazec na malé písmená
lstrip () Vráti verziu reťazca vľavo
maketrans () Vracia prekladovú tabuľku, ktorá sa použije v prekladoch
partition () Vracia n-ticu, kde je reťazec rozdelený na tri časti
lace () Vracia reťazec, kde je zadaná hodnota nahradená zadanou hodnotou
rfind () Vyhľadá reťazec pre zadanú hodnotu a vráti poslednú pozíciu, kde sa našiel
rindex () Vyhľadá reťazec pre zadanú hodnotu a vráti poslednú pozíciu, kde sa našiel
rjust () Vracia správne zarovnanú verziu reťazca
rpartition () Vráti n-ticu, kde je reťazec rozdelený na tri časti
rsplit () Rozdeľuje reťazec v určenom oddeľovači a vracia zoznam
rstrip () Vráti verziu reťazca s pravým orezom
split () Rozdeľuje reťazec v určenom oddeľovači a vracia zoznam
splitlines () Rozdeľuje reťazec pri zalomení riadku a vracia zoznam
beginwith () Vráti true, ak sa reťazec začína zadanou hodnotou
strip () Vracia orezanú verziu reťazca

swapcase () Swapcase, malé písmená sa stávajú veľké a naopak

title () Skonvertuje prvý znak každého slova na veľké písmená

translate () Vráti preložený reťazec

upper () Skonvertuje reťazec na veľké písmená

zfill () Na začiatok vyplní reťazec zadaným počtom 0 hodnôt

Metódy zoznamu Python / poľa

Python má sadu vstavaných metód, ktoré môžete použiť v zoznamoch / poliach.

append () Pridá prvok na koniec zoznamu

clear () Odstráni všetky prvky zo zoznamu

copy () Vráti kópiu zoznamu

count () Vráti počet prvkov so zadanou hodnotou

extend() Na koniec aktuálneho zoznamu pridajte prvky zoznamu (alebo iné iterovateľné položky)

index () Vracia index prvého prvku so zadanou hodnotou

insert () Pridá prvok na určené miesto

pop () Odstráni prvok na určenej pozícii

remove () Odstráni prvú položku so zadanou hodnotou

reverse () Obráti poradie zoznamu

sort () Zoraďuje zoznam

Metódy Python Dictionary

Python má sadu vstavaných metód, ktoré môžete použiť v slovníkoch.

clear () Odstráni všetky prvky zo slovníka

copy () Vráti kópiu slovníka

fromkeys () Vracia slovník so zadanými kľúčmi a hodnotami

get () Vracia hodnotu zadaného kľúča

items () Vracia zoznam obsahujúci n-ticu pre každý pár hodnôt kľúčov

keys () Vracia zoznam obsahujúci kľúče slovníka

pop () Odstráni prvok zadaným kľúčom

popitem () Odstráni posledný vložený pár kľúč - hodnota

setdefault () Vracia hodnotu zadaného kľúča. Ak kľúč neexistuje: vložte kľúč so zadanou hodnotou

update () Aktualizuje slovník pomocou zadaných párov kľúč - hodnota

value () Vráti zoznam všetkých hodnôt v slovníku

Pythonovy tuplové metódy

Python má dve vstavané metódy, ktoré môžete použiť na n-tice.

count () Vráti, koľkokrát sa zadaná hodnota vyskytuje v n-tici

index () Vyhľadá n-ticu pre zadanú hodnotu a vráti polohu, kde bola nájdená

Metódy súboru Python

Python má sadu vstavaných metód, ktoré môžete použiť v množinách.

add () Pridá prvok do množiny

clear () Odstráni všetky prvky zo sady

copy () Vracia kópiu sady

difference () Vracia množinu obsahujúcu rozdiel medzi dvoma alebo viacerými množinami

difference_update () Odstráni položky v tejto množine, ktoré sú tiež zahrnuté v inej špecifikovanej množine

discard () Odstrániť uvedenú položku

intersection () Vracia množinu, ktorá je priesečníkom dvoch ďalších množín
intersection_update () Odstráni položky v tejto množine, ktoré sa nenachádzajú v iných špecifikovaných množinách.
isdisjoint () Vracia, či dve sady majú priesečník alebo nie
issubset () Vracia, či táto sada obsahuje alebo nie
issuperset () Vracia, či táto sada obsahuje inú množinu alebo nie
pop () Odstráni prvok zo sady
remove () Odstráni určený prvok
symmetric_difference () Vracia množinu so symetrickými rozdielmi dvoch množín
symmetric_difference_update () vkladá symetrické rozdiely z tejto množiny a ďalších
union () Vráti množinu obsahujúcu spojenie množín
update () Aktualizujte súbor spojením tejto množiny a ďalších

Metódy súboru Python

Python má pre súborový súbor k dispozícii sadu metód.

close () Zatvorí súbor
detach () Vracia oddelený surový tok z vyrovnávacej pamäte
fileno () Vracia číslo, ktoré predstavuje tok, z pohľadu operačného systému

flush () Vyplachuje interný buffer

isatty () Vracia, či je tok súborov interaktívny alebo nie

read () Vráti obsah súboru

readable () Vráti, či sa tok súborov dá prečítať alebo nie

readline () Vracia jeden riadok zo súboru

readlines () Vracia zoznam riadkov zo súboru

seek () Zmena umiestnenia súboru

seekable () Vracia, či nám súbor umožňuje zmeniť pozíciu súboru

tell () Vráti aktuálnu pozíciu súboru

truncate () Zmení veľkosť súboru na zadanú veľkosť

writeable () Vráti, či sa dá súbor zapísať alebo nie

write () Zapíše zadaný reťazec do súboru

writelines () Zapíše zoznam reťazcov do súboru

Python Kľúčové slová

Python má množinu kľúčových slov, ktoré sú vyhradené slová, ktoré nemožno použiť ako názvy premenných, názvy funkcií alebo iné identifikátory:

and logický operátor

as Vytvorenie aliasu

assert Na ladenie

break Prelomiť slučku

class Definovanie triedy

continue Pokračovanie do ďalšej iterácie slučky

def Definovanie funkcie

del Vymazanie objektu

elif Používa sa v podmienených príkazoch, rovnako ako v iných prípadoch
else Používa sa v podmienených príkazoch
except Používa sa s výnimkami, čo robiť, keď sa vyskytne výnimka
False booleovská hodnota "nepravda", výsledok porovnávacích operácií
finally Použitý s výnimkami, blok kódu, ktorý bude vykonaný bez ohľadu na to, či existuje výnimka alebo nie
for Vytvorenie slučky for for
from Na importovanie konkrétnych častí modulu
global Vyhlásenie globálnej premennej
if urobiť podmienečné vyhlásenie
import Ak chcete importovať modul
in Ak chcete skontrolovať, či je hodnota v zozname, v tuple atď.
is Testovať, či sú dve premenné rovnaké
lambda Na vytvorenie anonymnej funkcie
None Predstavuje nulovú hodnotu
nonlocal Na deklarovanie nelokálnej premennej
not logický operátor
or Logický operátor
pass Nulove vyhlásenie, vyhlásenie, ktoré neurobí nič
raise Zvýšenie výnimky
return Na ukončenie funkcie a vrátenie hodnoty
True booleovská hodnota pravda, výsledok porovnávacích operácií

try skúsiť ... okrem vyhlásenia

while Vytvorenie slučky while

with Používa sa na zjednodušenie spracovania výnimiek

yield Na ukončenie funkcie vráti generátor

Ako odstrániť duplikáty zo zoznamu Python

Naučte sa, ako odstrániť duplikáty zo zoznamu v Pythone.

príklad

Odstráňte duplikáty zo zoznamu:

```
mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist)
```

```
['a', 'b', 'c']
```

Najprv máme zoznam, ktorý obsahuje duplikáty:

Zoznam s duplikátmi

```
mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist)
```

Vytvorte slovník pomocou položiek zoznamu ako kľúčov. Týmto sa automaticky odstránia všetky duplikáty, pretože slovníky nemôžu mať duplicitné kľúče.

Vytvorte slovník

```
mylist = ["a", "b", "a", "c", "c"]
mylist = list( dict.fromkeys(mylist) )
print(mylist)
```

Potom skonvertujte slovník späť do zoznamu:

Konvertovať do zoznamu

```
mylist = ["a", "b", "a", "c", "c"]
mylist = list( dict.fromkeys(mylist) )
print(mylist)
```

Teraz máme zoznam bez duplikátov a má rovnaké poradie ako pôvodný zoznam.

Vytlačte zoznam a ukážte výsledok

Vytlačte zoznam

```
mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist)
```

Vytvorte funkciu

Ak chcete mať funkciu, kde môžete posielať zoznamy a získať ich späť bez duplikátov, môžete vytvoriť funkciu a vložiť kód z vyššie uvedeného príkladu.

príklad

```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

```
['a', 'b', 'c']
```

Vytvorte funkciu, ktorá ako argument vezme zoznam.

Vytvorte funkciu

```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

Vytvorte slovník pomocou týchto položiek zoznamu ako kľúčov.

Vytvorte slovník

```
def my_function(x):
  return list( dict.fromkeys(x) )

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

Previesť slovník do zoznamu.

Konvertovať do zoznamu

```
def my_function(x):
  return list( dict.fromkeys(x) )

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

Vrátiť zoznam

Návratový zoznam

```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

Vyvolajte funkciu so zoznamom ako parametrom:

Zavolajte funkciu

```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

Vytlačiť výsledok:

Vytlačte výsledok

```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

Ako prevrátiť reťazec v Pythone

Naučte sa, ako zmeniť reťazec v Pythone.

Neexistuje zabudovaná funkcia na zmenu reťazca v Pythone.

Najrýchlejší (a najjednoduchšie?) Spôsob ich použiť plátok že kroky dozadu, `-1`.

príklad

Obráťte reťazec „Ahoj svet“:

```
txt = "Hello World"[::-1]
print(txt)
```

```
dlroW olleH
```

Máme reťazec „Ahoj svet“, ktorý chceme zvrátiť:

Reťazec zvrátiť

```
txt = "Hello World" [::-1]
print(txt)
```

Vytvorte rez, ktorý sa začína na konci reťazca a posúva dozadu.

V tomto konkrétnom príklade `[::-1]`znamená výrez plátok začiatok na konci reťazca a koniec v polohe 0, pohybuje sa krokom `-1`, *záporný* , čo znamená jeden krok dozadu.

Rozdeľte reťazec

```
txt = "Hello World" [::-1]
print(txt)
```

Teraz máme reťazec, `txt`ktorý znie „Ahoj svet“ dozadu.

Vytlačte reťazec a ukážte výsledok

Vytlačte zoznam

```
txt = "Hello World"[::-1]
print(txt)
```

Vytvorte funkciu

Ak chcete mať funkciu, kam môžete poslať svoje reťazce a vrátiť ich späť, môžete vytvoriť funkciu a vložiť kód z vyššie uvedeného príkladu.

príklad

```
def my_function(x):
  return x[::-1]
```

```
mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```

```
sdrawkcab ekil skool txet siht woh rednow I
```

Vytvorte funkciu, ktorá ako argument berie reťazec.

Vytvorte funkciu

```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```

Reťazec nakrájajte na začiatok struny a posuňte sa dozadu.

Rozdeľte reťazec

```
def my_function(x):
  return x [::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```

Vráťte späť reťazec

Vráťte reťazec

```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt )
```

Vyvolajte funkciu s reťazcom ako parametrom:

Zavolajte funkciu

```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```

Vytlačiť výsledok:

Vytlačte výsledok

```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```