Въведение в Python 1

Какво е Python

Python е интерпретируем, интерактивен, обектно-ориентиран език за програмиране с отворен код, създаден от Гуидо ван Росум в началото на 90-те години. Кръстен е на телевизионното шоу на BBC „Monty Python's Flying Circus".

Идеята за Python се заражда в края на 1980-те, като реалното осъществяване започва през декември 1989 г. от Гуидо ван Росум в CWI (Centrum Wiskunde & Informatica – международно признат изследователски институт по математика и компютърни науки, базиран в Амстердам, Холандия).

Python позволява разделянето на една програма на модули, които могат да се използват отново в други програми. Също така притежава голям набор от стандартни модули, които да се използват като основа на програмите. Съществуват и вградени модули, които обезпечават такива неща като файлов вход/изход (I/O), различни системни функции, сокети (sockets), програмни интерфейси към GUI-библиотеки като Tk, както и много други.

Идентификатори в Python

Идентификаторите в Python започват с буква или долно тире, съдържат латински главни и/или малки букви, долно тире и цифри, и са неограничени по дължина.

In [ ]:
```
a_b = 10 #snake case
ab = 20

print(a_b)
print(ab)
```

```
10
20
```

Основни типове данни

`int, float, long, complex, bool, str, chr, ord`

В Python типа на операндите не се заявява предварително. Ключовите думи: int, float, long, comlex, bool и str се използват за преобразуване от един тип към друг.

In [ ]:
```
num_str = "10"
num = 20

print(num + int(num_str))
print(str(num) + num_str)

#error
print(num + num_str)
```

```
30
2010
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-38f1e1cc0b19> in <module>
      6
      7 #error
----> 8 print(num + num_str)

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

Коментар

За да коментирате ред в python се използва знака #

Свързване на два или повече реда

За свързване на два или повече реда, се използва наклонена черта \:

```
if a >= b and a<=c:
  print("a is between b and c")
  a=b

if a >= b and \
   a <= c:
  print("a is between b and c")
```

При създаване на списък, отделните елементи могат да се въвеждат на няколко реда. Коментари могат да се добавят на всеки ред.

In [ ]:
```
a = 20
b = 10
c = 30

if a >= b and \
   a <= c:
  print("a is between b and c")

if (a >=b and   #check for a>=b
    a <=c):
  print("a is between b and c")
```

```
a is between b and c
a is between b and c
```

In [ ]:
```
arr = [ 1, 2, 3,  # comment
        4, 5,
        6]
print(arr)
```

```
[1, 2, 3, 4, 5, 6]
```

Аритметични операции

`+, -, * , **, /, //, %`

Примери:

In [ ]:
```
print(2 ** 3)  # степенуване

print(11 / 2)  # деление

print(11 // 2) # целочислено деление

print(11 % 2)  # остатък от деление
```

```
8
5.5
5
1
```

Оператори за сравнение

`<` `>` `<=` `>=` `==` `!=`

Съставни оператори

```
+=   -=   *=   /=   //=  %=   @=
&=   |=   ^=   >>=  <<=  **=
```

In [ ]:
```
Дена = 10

#wrong
#a++
a+=1 # a = a + 1
```

Логически оператори

`and` `or` `not`

Оператори за вход и изход

Оператор за вход

`input([prompt]) -> string`

Чете низ от стандартния вход

Примери:

In [ ]:
```
# Input string
name = input("Enter your name: ")

name
```

```
Enter your name: George
'George'
```
Out[ ]:

In [ ]:
```
# Input number
num = int(input("Enter number: "))

num
```

```
Enter number: 10
10
```
Out[ ]:

In [ ]:
```
# Input float number
float_num = float(input("Enter float number: "))

float_num
```

```
Enter float number: 10.5
10.5
```
Out[ ]:

Оператор за изход

```
print(value, ...., sep=' ', end='\n')
```

In [ ]:
```
num = 10

#print multiple things
print("number", num, sep=",")

#print without new line
print("10,", end="")
print("20")

#printing with formatting
print("%s %d" % ("number",2))

#printing with formatting 2
print("{0} {1} {1}".format("string",100))

#printing with f-Strings:
print(f"Our number is {num}")
```

```
number,10
10,20
number 2
string 100 100
Our number is 10
```

Условен уператор

```
if condition:      # : required
  operators        # ident required
elif condition2:   # : required
  operators        # ident required
else:              # : required
  operators        # ident required
```

In [ ]:
```
a = 10
b = 20
c = 30

if a >= b:
  print("a>=b")
  print("-----")
elif c > b:
  print("a>b")
elif c == b:
  pass
else:
  print("something else")

#uses mathematical order
if (c>=b>=a):
  print("c>=b>=a")
```

Списъци

(List) - съставен тип данни за групиране на стойности, от различен и/или един и същи тип, изписани като поредица и разделени със запетая между квадратни скоби. Достъпът до елемент се осъществява чрез индекс, цяло число. Дължината на всеки списък може да се променя и да не се определя предварително. Списъците

могат да се влагат един в друг.

Примери:

```
# empty list
my_list = []
my_list = list()

# list of integers
my_list = [1, 2, 3]

# list with mixed data types
my_list = [1, "Hello", 3.4]

# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Достъп до елементите на списък

Има различни методи за достъп до елементите на списъците

Индексиране

Достъпа до определен елемент в списъка чрез неговия индекс става с оператора `[]`. Индексите започват от `0`. Ако един списък има `5` елемента индексите ще са между `0` и `4` включително.

Опит да се достъпи индекс извън границите на списъка ще предизвика `IndexError`. Вложени списъци може да се достъпват чрез вложено индексиране `[y][x]`

In [ ]:
```
# List indexing

my_list = ['p', 'r', 'o', 'b', 'e']

# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Nested List
n_list = ["Happy", [2, 0, 1, 5]]

# Nested indexing
print(n_list[0][1])

print(n_list[1][3])
```

```
p
o
e
e
a
5
```

Python позволява отрицателни индекси. Индекс -1 сочи към последния елемент, индекс -2 сочи към пред-последния и т.н.

In [ ]:
```
# Negative indexing in lists
my_list = ['p','r','o','b','e']

print(my_list[-1])

print(my_list[-5])
```

```
e
p
```

Изрязване на списъци (list slicing)

Достъп до диапазон от елементи в списъците (под-списък) се осъществява с оператора `:`

In [ ]:
```
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])
```

```
# elements 6th to end
print(my_list[5:])
print(my_list[5:len(my_list)])

# elements beginning to end and create a copy
print(my_list[:])
print(my_list[-9:len(my_list)])
```

```
['o', 'g', 'r']
['p', 'r', 'o', 'g']
['a', 'm', 'i', 'z']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

Добавяне и промяна на елементи

Списъците могат да бъдат променяни чрез оператора `=`

In [ ]:
```
# Correcting mistake values in a list
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

print(odd)
```

```
[1, 4, 6, 8]
[1, 3, 5, 7]
```

Добавяне на един елемент в списък става с `append()` метод, а добавяне на много елементи едновременно става с `extend()` метод.

In [ ]:
```
# Appending and Extending lists in Python
odd = [1, 3, 5]

odd.append(7)

print(odd)

odd.extend([9, 11, 13])

print(odd)
```

```
[1, 3, 5, 7]
[1, 3, 5, 7, 9, 11, 13]
```

Също така оператор `+` се използва за обединяване на списъци, а `*` за повторение на списъка няколко пъти.

In [ ]:
```
# Concatenating and repeating lists
odd = [1, 3, 5]

print(odd + [1])

print(odd + [9, 7, 5])

print(["re", "re1"] * 3)
```

```
[1, 3, 5, 1]
[1, 3, 5, 9, 7, 5]
['re', 're1', 're', 're1', 're', 're1']
```

Премахване на елементи от списък

Изтриване на елементи или дори целия списък може да се прави с специалния оператор `del`.

In [ ]:
```
# Deleting list items
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# delete one item
del my_list[2]

print(my_list)

# delete multiple items
del my_list[1:5]

print(my_list)
```

```
# delete entire list
del my_list

# Error: List not defined
print(my_list)
```

```
['p', 'r', 'b', 'l', 'e', 'm']
['p', 'm']
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-37-081d50d030eb> in <module>
     16
     17 # Error: List not defined
---> 18 print(my_list)

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

Други методи за премахване на елементи са `remove()` метод за премахване на елемент по индекс, `pop()` за изваждане на последния елемент от списъка (може да се ползва за използването на списъка като стек). `clear` служи изчисване на списъка.

In [ ]:

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```

```
['r', 'o', 'b', 'l', 'e', 'm']
o
['r', 'b', 'l', 'e', 'm']
m
['r', 'b', 'l', 'e']
[]
```

Други методи поддъжани от списъците

In [ ]:

```
# Python list methods
my_list = [3, 8, 1, 6, 0, 8, 4]

# find element
# Output: 1
print(my_list.index(8))

# count element occurances
# Output: 2
print(my_list.count(8))

# sort list
my_list.sort()
# return new sorted list
sorted_list = sorted(my_list)

# Output: [0, 1, 3, 4, 6, 8, 8]
print(my_list)

#reverse the list elements
my_list.reverse()

# Output: [8, 8, 6, 4, 3, 1, 0]
print(my_list)
```

```
1
2
[0, 1, 3, 4, 6, 8, 8]
[8, 8, 6, 4, 3, 1, 0]
```

List Comprehension: елегантен начин да се създават списъци

List Comprehension е елегантен начин за създаване на списъци от други списъци. List Comprehension е начин при

който се влага `for` израз вътре в квадратни скоби за списък `[]`.

Това е пример за създаване на списък със степени на 2:

In [ ]:
```
pow2 = [2 ** x for x in range(10)]
print(pow2)
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Този код е еквивалентен на:

In [ ]:
```
pow2 = []
for x in range(10):
   pow2.append(2 ** x)
print(pow2)
```

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

List Comprehension може също да съдържа множество `for` и също така `if` оператори

In [ ]:
```
pow2 = [2 ** x for x in range(10) if x > 5]
print(pow2)

odd = [x for x in range(20) if x % 2 == 1]
print(odd)
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

print([x+y for x in ['Python ','C '] for y in ['Language','Programming']])
```

```
[64, 128, 256, 512]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
['Python Language', 'Python Programming', 'C Language', 'C Programming']
```

Прочитане на множество елементи от стандартния вход

In [ ]:
```
#reading numbers from the standard input, spearated by space
numbers = [int(n) for n in input().split()]
# numbers = [int(n) for n in input().split(' ')]
# numbers = [int(n) for n in input().split(', ')]
print(numbers)
```

```
1 2 3 4 5
[1, 2, 3, 4, 5]
```

Брой елементи в списък

Броя елементи в списък може да се вземе чрез `len()` метода.

In [ ]:
```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']
print(len(my_list))
```

```
7
```

Комплект (tuple)

Комплекта се състои от елементи, които могат да бъдат произволни: прости константи или обекти. Елементите на комплекта не е задължително да са еднородни.

Комплектите (подобно на символните низове) са непроменливи: не е възможно да се присвои стойност на даден елемент от комплекта. Възможно е създаването на комплект, който съдържа в себе си променливи обекти – например списък (list).

Комплектите могат да бъдат вложени един в друг. Дефинират се чрез изброяване със запетайка, поставени в кръгли скоби.

In [ ]:
```
#empty tuple
t=()
# t = tuple()
# t = (1,)

#tuple with 3 elements
t=(1,2,3)

#tuple with mixed elements
t=("hello", 2, 3)

#tuple length
print(len(t))

#set multiple variables at once
a,b = 1,2
```

```
print(a)
print(b)

#tuple unpacking
y = (10,20,30)
a,b,c = y
#a,b,c = func()
#a,b,_ = func()
#def func():
#  return (10,20,30)

#access 2nd element in a tuple
print(y[1])

#tuples are immutable. this is wrong
#y[1] = 10
```

```
3
1
2
20
```

Речник (dictionary/map)

Речникът е неподредено множество от двойки (ключ, стойност), като ключа е константа от всеки тип (най-вече символни низове). Комплекти могат да бъдат използвани за ключове само в случай, че не използват променливи обекти. Не може да се използва списък (list), тъй като той притежава възможността да се променя.

Основните операции са:

- Съхраняване на стойност под някакъв ключ (съхраняването на две стойности под един и същи ключ е невъзможно, запазва се втората такава подадена със съответния ключ).

In [ ]:
```
#empty dictionary
p = dict()
#p={}

#set single dictionary entry key="name", value="John"
p["name"] = "John"
p["age"] = 20

print(p)

#initialize dictionary
p = {"name": "John", "age": 20}
```

```
{'name': 'John', 'age': 20}
```

- Извличане на стойността при подаване на ключа

In [ ]:
```
p = {"name": "John", "age": 20}

#access values with key
print("Name: ", p["name"])
print("Age: ", p["age"])
```

```
Name:  John
Age:  20
```

- Премахване на двойка ключ – стойност чрез „del“

In [ ]:
```
p = {"name": "John", "age": 20}

#delete key/value
del p["age"]

print(p)
```

```
{'name': 'John'}
```

- Методът keys() на обект от тип речник връща списък с всички използвани ключове

In [ ]:
```
p = {"name": "John", "age": 20}

print(p.keys())

print(p.values())
print(p.items())
```

```
dict_keys(['name', 'age'])
dict_values(['John', 20])
dict_items([('name', 'John'), ('age', 20)])
```

Оператори за членство

Операторите `in` и `not in` се използват за проверка дали дадена стойност или променлива се съдържа в дадена последователност - string, list, tuple, set и dict.

In [ ]:
```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)

# Output: True
name = "Ivan"
print("an" in name)
```

Циклични оператори

```
while condition:     # : required
  operators          # ident required
for x1 in x2:        # : required
  operators          # ident required
```

където:

x1 - управляваща променлива

x2 - списък, редица или низ

- С оператора **break** се осъществява излизане от цикъл
- Оператора **continue** прекъсва текущата итерация и продължава изпълнението на цикъла от следващия елемент

In [ ]:
```
#fibonacci sequence
a, b = 0, 1
n = 10
while b < n:
  print(b)
  a, b = b, a+b
```

```
1
1
2
3
5
8
```

Функция **range()** - генерира списъци, съдържащи аритметични прогресии

Синтаксис:

range(stop)

или

range(start, stop[, step])

In [ ]:
```
for i in range(5):
  print(i)

print("--------------------")

for i in range(0,10,2):
  print(i)

print("--------------------")

s="abcdef"
for i in range(len(s)):
  print(s[i])

print("--------------------")
```

```
for i in range(len(s)):
  print(s[i], end='')
print()

print("--------------------")

for ch in s:
  print(ch)

for fruit in ['apple','banana','mango']:
  print("I like",fruit)

#loop n times without using the index
for _ in range(10):
  print("a", end="")
```

```
0
1
2
3
4
--------------------
0
2
4
6
8
--------------------
a
b
c
d
e
f
--------------------
abcdef
--------------------
a
b
c
d
e
f
I like apple
I like banana
I like mango
aaaaaaaaaa
```

Математически функции

В Python има различни математически функции. За да може да се използват трябва да се използва библиотека math по следния начин:

```
import math
```

In [ ]:

```
import math

#square root
print( math.sqrt(9) )

#e^x
print(math.exp(1))

#logarithm
print(math.log(1))

#logarithm base 2
print(math.log2(8))

#logarighm base 10
print(math.log10(100))

#power
print(math.pow(2,4))

#factorial
print(math.factorial(4))
```

```
3.0
2.718281828459045
0.0
3.0
2.0
16.0
24
```

Символни низове (string)

Символни низове (String)

Символните низове в Python могат да са затворени в единични или двойни кавички

```
s = "hello"
```

```
s1 = 'hello'
```

Символните низове не могат да бъдат променяни (immutable). Всички операции създават нов низ.

Специални символи в низовете:

In [ ]:
```
# new line
print("Hello\nWorld")

#empty line
print()

# tab
print("Hello\tWorld")
```

```
Hello
World

Hello	World
```

Различни операции с низове:

In [ ]:
```
#Concatenate strings
print('ab' + "cd")

#Indexing letters in a string

s = "Hello World!"

#Wrong:
#s[0]="a"

#single element indexing
print(s[4])

#slicing strings
print(s[1:3])

#using negative indexes
print(s[-2])

#show first 5 letters
print(s[:5])

#show last 6 letters
print(s[-6:])

#string length
print(len(s))

#count occurances of substring
print(s.count("ll"))

#Convert to upper case
print(s.upper())

#Convert to lower case
print(s.lower())

#Split the string in substrings with a separator
print(s.split())
print(s.split("r"))

#replace substring with another string
print(s.replace(" ","-"))

#looping over the letters of a string
for ch in s:
  print(ch)
```

```
abcd
o
el
d
Hello
World!
12
1
HELLO WORLD!
hello world!
['Hello', 'World!']
['Hello Wo', 'ld!']
Hello-World!
H
e
l
l
o

W
o
r
l
d
!
```