

Машинно обучение

Лабораторно упражнение №1

Въведение в Python

Python е език от високо ниво, който позволява програмата да се раздели на модули, които могат да бъдат използвани, отново в други програми на Python. Езикът съдържа набор от вградени модули, които обезпечават различни функции като: файлов вход/изход (I/O), системни функции, сокети (sockets), програмни интерфейси към GUI библиотеки.

Python е интерпретируем език, като не са необходими компилиране и свързване. Интерпретаторът може да се използва интерактивно, което го прави лесен за експериментиране с възможностите на езика, или за тестване на функциите по време на разработка отдолу-нагоре.

Езикът е кръстен на шоуто на BBC "Monty Python's Flying Circus".

1. Интерактивен режим.

В интерактивен режим промпта представлява три знака по-голямо (>>>), а за продължение на ред се появява вторичен промпт, обикновено три точки (...). В този режим Python може да се използва като калкулатор. В интерактивен режим на Spyder първия промпт е `In [1]:`, като числото в квадратни скоби инициира номера на командния ред, на който пишете.

Помощна информация за елементите на езика, може да получите като въведете `help()`. Появява се пояснителен текст и от появилия се списък, въвеждате търсения елемент.

2. Идентификатори в Python. – идентификаторите започват с буква или долно тире, съдържат латински главни и/или малки букви, долно тире и цифри, и са неограничени по дължина. Запазени идентификатори са тези идентификатори, които започват със следните символи: `_*`, `_*_`, `__*`.

3. Ключови думи в Python. – ключовите думи са запазени за езика и не могат да се използват като обикновени идентификатори.

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

4. Основни типове данни.

```
int, float, long, complex, bool
```

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

5. **Коментар.** – използва се знака #

6. **Свързване на два или повече физически реда.** – използва се наклонена черта \

Пример 1 :

```
x=[1,\n    2]
```

x

Резултат: ?

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

Пример 2 :

```
y=[1,2,# коментар\n    3,4,5,\n    6,7]
```

y

Резултат: ?

7. Оператори.

+	-	*	**	/	//	%
<<	>>	&		^	~	
<	>	<=	>=	==	!=	

8. Разделители.

()	[]	{	}		
,	:	.	;	@	=	->	
+=	--	*=	/=	//=	%=	&=	=
^=	>>=	<<=	**=				

9. Аритметични операции. – +, -, *, **, /, //, %

Пример 3:

```
2**3 # степенуване
```

Резултат: ?

Пример 4 :

9/2 # деление

Резултат: ?

Пример 5 :

9//2 # целочислено деление

Резултат: ?

Пример 6 :

9%2 # остатък от деление

Резултат: ?

10. Оператори за сравнение.

< > <= >= == !=

11. Побитови операции.

<< >> & | ^ ~

12. Съставни оператори.

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

13. Логически оператори.

and or not

14. Оператори за вход и изход.

14.1. Оператор за вход.

`input([prompt]) -> string`

Чете низ от стандартния вход

Пример 7:

```
name=input("Your name:")
```

```
Your name: (Примерно: Ana)
```

```
name
```

Резултат: ?

Пример 8:

```
number=input("Enter integer:")  
Enter integer: (Примерно: 3)  
number
```

Резултат: '3'

В примера number е string.

Пример 9:

```
number=int(input("Enter integer:"))  
Enter integer: (Примерно: 3)  
number
```

Резултат: 3

В примера number е цяло число.

14.2. Оператор за изход.

```
print(value, ..., sep=' ', end='\n', file=sys.stdout,  
flush=False)
```

За повече информация help(print).

Пример 10:

```
print("number=", number)
```

Резултат: ?

15. Списък. (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.

Пример 11:

```
ime='Ivan'  
familia='Ivanov'  
student=['ime:', ime, 'familia:', familia, 4, 'kurs']  
student
```

Резултат: ?

15.1. Индексиране.

Пример 12:

```
student[0]
```

Резултат: ?

Пример 13:

```
student[-1] # извежда последния елемент
```

Резултат: ?

Пример 14:

```
student[0:2] # извежда елементи от индекс 0 до индекс 1
```

Резултат: ?

Пример 15:

```
student[:3] # извежда първите три елемента
```

Резултат: ?

Пример 16:

```
student[3:] # извежда всичко, освен първите три елемента
```

Резултат: ?

15.2. Замяна на елементи.

Пример 17:

```
student[1]='Dian'
```

```
student
```

Резултат: ?

15.3. Конкатенация на списъци.

Пример 18:

```
fnomer=['fN',123456]
```

```
student=student + fnomer
```

```
student
```

Резултат: ?

15.4. Изтриване на елементи.

Пример 19:

```
student[6:]=[ ]
```

```
student
```

Резултат: ?

15.5. Създаване на празен списък.

Пример 20:

```
student1=list()
```

```
student1
```

Резултат: ?

15.6. Визуализация на списък.

Пример 21:

```
list(student)
```

Резултат: ?

15.7. Списъкът може да се третира и като двумерен масив.

Пример 22:

```
student[1][0]
```

Резултат: ?

15.8. Списъци. Методи.

15.8.1. Разширяване на списък чрез extend.

Пример 23:

```
student.extend(fnomer)
```

```
student
```

Резултат: ?

Но:

Пример 24:

```
student.extend('TU-Varna')
```

```
student
```

Резултат: ?

15.8.2. Разширяване на списък чрез append.

Пример 25:

```
student.append('TU-Varna')
```

```
student
```

Резултат: ?

Но:

Пример 26:

```
year=['year',2018]
student.append(year)
student
```

Резултат: ?

15.8.3. Преброяване на повтарящите се елементи.

Пример 27:

```
student1=['Koev', 2,2,3,4,5,5,5]
student1.count(5)
```

Резултат: ?

15.8.4. Индекси на списък.

L.index(value, [start, [stop]]) -> integer -- return first index of value.

Пример 28:

```
student1.index('Koev')
```

Резултат: ?

Пример 29:

```
student1.index(2,2,7)
```

Резултат: ?

Самостоятелна задача 1:

Разгледайте самостоятелно методите: **clear, insert, pop, remove, reverse, sort**. За повече подробности: `help(list)`. Тествайте със собствени примери.

16. Комплекти. - "tuple"

Компектите са съставен тип данни за групиране на стойности, от различен и/или един и същи тип, изписани като поредица и разделени със запетая и без/със заграждащи скоби

Пример 30:

```
x='hi'
y=x,'world',2018
y
```

Резултат: ?

В резултата комплектите винаги са затворени в скоби

16.1. Празен комплект.

Пример 31:

```
c=()
```

16.2. Комплект с един елемент.

Пример 32:

```
d=1,  
d
```

Резултат: ?

16.3. Дължина на комплект.

Пример 33:

```
len(d)
```

Резултат: ?

16.4. Присвояване на стойности на повече от една променлива.

Пример 34:

```
i,j=1,2  
i
```

Резултат: ?

Пример 35:

```
j
```

Резултат: ?

16.5. Извличане на стойности от комплекта – разкплектоване.

Пример 36:

```
a,b,c=y  
a,b,c
```

Резултат: ?

Пример 37:

```
a
```

Резултат: ?

Пример 38:

```
b
```


Резултат: ?

Пример 39:

c

Резултат: ?

16.6. Едновременно създаване на комплект и сравняване.

Пример 40:

1,2 < 1,3

Резултат: ?

16.7. Сравняване на комплекти.

Сравнява първия елемент от първия комплект с първия елемент от втория комплект, ако е лъжа сравнява следващата двойка елементи.

Пример 41:

(4, 5) < (3, 5)

Резултат: ?

Пример 42:

(2, 6) < (3, 5)

Резултат: ?

16.8. Комплекти. Методи.

16.8.1. Преброяване на повтарящите се елементи.

Пример 43:

y.count(2018)

Резултат: ?

16.8.2. Индекси на елемент от комплект.

Пример 44:

y.index(2018)

Резултат: ?

17. Оператори за членство. - използват се за проверка дали дадена стойност или променлива се съдържа в дадена последователност - string, list, tuple, set и dict.

Оператор	Значение	Пример
in	True, ако стойността/променливата е намерена в дадената последователност - string, list, tuple, set и dictionary	2 in x
not in	True, ако стойността/променливата не е намерена в дадената последователност - string, list, tuple, set и dictionary	2 not in x

Пример: Проверка дали даден елемент се съдържа в списък - със служебната дума *in*

Пример 45:

```
x=[1,2,3]
```

```
2 in x
```

Резултат: ?

18. Оператори за идентичност. - използват се за проверка дали две променливи сочат към една и съща клетка от паметта. Две променливи, които са равни, може да са идентични, може и да не са.

Оператор	Значение	Пример
is	True, ако операндите са идентични	x is y
is not	True, ако операндите не са идентични	x is not y

17.1. Променливите са от цял тип.

Пример 46:

```
u=9
```

```
v=9
```

```
u==v
```

Резултат: ?

Пример 47:

```
u is v
```

Резултат: ?

Пример 48:

```
id(u), id(v)
```

Резултат: ?

Извод: Стойностите им са както равни, така и идентични.

И двете променливи сочат към една и съща клетка от паметта. Идентификационните им номера са равни. За повече информация `help(id)`.

Самостоятелна задача 2:

Проверете идентични ли са 2 променливи, които съдържат еднакви низове.

Какъв е извода?

17.2. Променливите са от реален тип. Стойностите им са равни, но не са идентични.

Пример 49:

```
a=1.234
```

```
b=1.234
```

```
a==b
```

Резултат: ?

Пример 50:

```
a is b
```

Резултат: False

Пример 51:

```
id(a), id(b)
```

Резултат: ?

Но, ако свържем двете променливи със знак за равенство

Пример 52:

```
a=b
```

```
a is b
```

Резултат: True

Пример 53:

```
id(a), id(b)
```

Резултат: ?

Извод: Стойностите на двете променливи от реален тип след свързване със знака за равенство са равни и идентични.

Самостоятелна задача 3:

Проверете идентични ли са 2 променливи, които съдържат еднакви списъци от елементи.

Какъв е извода?

19. Условни оператори.

```
if (условен израз):      # двете точки са задължителни
    оператор             # задължително трябва да има отстъп
elif:                   # двете точки са задължителни
    оператор             # задължително трябва да има отстъп
else:                   # двете точки са задължителни
    оператор             # задължително трябва да има отстъп
```

20. Циклични оператори.

20.1. Оператор за цикъл while.

```
while (условен израз):  # двете точки са задължителни
    оператор 1           # задължително трябва да има отстъп
    оператор 2           # задължително трябва да има отстъп
    . . .
    Оператор n          # задължително трябва да има отстъп
```

20.2. Оператор за цикъл for.

```
for x1 in x2:           # двете точки са задължителни
    оператор 1           # задължително трябва да има отстъп
    оператор 2           # задължително трябва да има отстъп
    . . .
    Оператор n          # задължително трябва да има отстъп
```

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

21. Функция range(). – генерира списъци, съдържащи аритметични прогресии

Синтаксис: `range(stop)` или `range(start, stop[, step])`

За повече подробности: `help(range)`

Примери:

Пример 54:

```
range(5)
```

Резултат: ?

Това са числата от 0 до 4 включително, т.е. 0,1,2,3,4

Пример 55:

```
for i in range(5):  
    print(i,end=" ")
```

Резултат: ?

Пример 56:

```
for i in range(0,10,2):  
    print(i,end=" ")
```

Резултат: ?

Често операторът range се комбинира с оператора len.

Пример 57:

```
x="ab"  
for i in range(len(x)):  
    print(x[i])
```

Резултат: ?

22. Математически функции. – чрез добавяне на библиотеката `math`

Примери:

Пример 58:

```
import math
```

Пример 59:

```
math.sqrt(9)
```

Резултат: ?

Пример 60:

```
math.exp(1)
```

Резултат: ?

Пример 61:

```
math.log(1)
```

Резултат: ?

Пример 62:

```
math.log(math.exp(1))
```

Резултат: ?

Пример 63:

```
math.log2(8)
```

Резултат: ?

Пример 64:

```
math.log10(100)
```

Резултат: ?

Пример 65:

```
pow(2, 4)
```

Резултат: ?

Пример 66:

```
math.factorial(4)
```

Резултат: ?

За повече информация: `help(math)`

23. Работа с комплексни числа.

23.1. реална част - `z.real`

23.2. имагинерна част - `z.imag`

Примери:

Пример 67:

```
z=8+3j
```

```
z.real
```

Резултат: ?

Пример 68:

```
z.imag
```

Резултат: ?

24. Самостоятелни задачи.

24.1. Самостоятелна задача 4:

С помощта на оператора while или for, изчислете и изведете числата на Фибоначи.

Отворете нов файл от File→New File. Запишете файла като име_на_файл.py. Стартирайте го от Run→Run Module или F5.

24.2. Самостоятелна задача 5:

Създайте програма на Python, която да проверява дали въведена дума от потребител е палиндром.

24.3. Самостоятелна задача 6:

Създайте програма на Python, която да преброява гласните във въведена дума от потребител.

Използвайте функциите lower() и count().

25. Полезни връзки.

25.1. <https://docs.python.org/3/>

25.2. <https://www.programiz.com/python-programming>

25.3. <https://www.tutorialspoint.com/python/index.htm>

25.4. <https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ch1.html>