Python programming for beginners

Stefan Zhauryd
Instructor

Module 3

Boolean Values, Conditional Execution, Loops, Lists and List Processing, Logical and Bitwise Operations

In this module, you will learn about:

- **multidimensional lists and their applications.**

```
a = [0] * 5
print(a)

b = [0 for i in range(5)]
print(b)

c = [i*i for i in range(5)]
print(c)

d = [int(i) for i in input().split()]
print(d)
```

```
>>>
======= RESTART: D:/Pyt
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 1, 4, 9, 16]
2
[2]
>>>
======= RESTART: D:/Pyt
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 1, 4, 9, 16]
2 4 5 8 1 1 0
[2, 4, 5, 8, 1, 1, 0]
>>>
```

Lists in lists

```
1  squares = [x ** 2 for x in range(10)]
2  odds = [x for x in squares if x % 2 != 0 ]
3
4  print(odds)
```

Console >_

```
[1, 9, 25, 49, 81]
```

```
1  squares = [x ** 2 for x in range(10)]
2
3  print(squares)
```

Console >_

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
1  twos = [2 ** i for i in range(8)]
2
3  print(twos)
```

Console >_

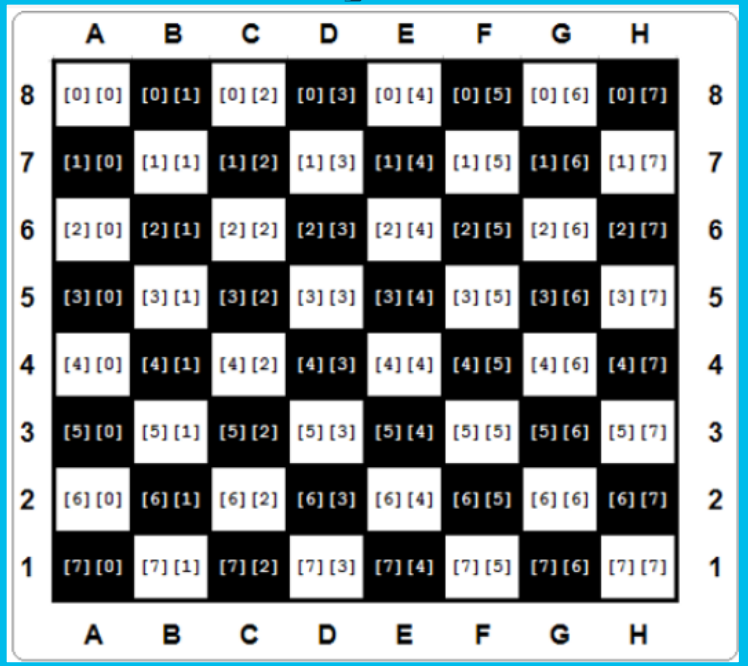
```
[1, 2, 4, 8, 16, 32, 64, 128]
```

Lists in lists: two-dimensional arrays - continued

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(a)
print(a[1])
print(a[2])
print(a[0])
print()


print(a[0][0])
print(a[0][1])
print(a[0][2])
print()

print(a[2][0])
print(a[2][1])
print(a[2][2])
print()
```

```
====== RESTART: D:/Python content
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[4, 5, 6]
[7, 8, 9]
[1, 2, 3]

1
2
3

7
8
9

>>>
```

Lists in lists: two-dimensional arrays - continued




```
EMPTY = "-"
ROOK = "ROOK"
board = []

for i in range(8):
    row = [EMPTY for i in range(8)]
    board.append(row)

board[0][0] = ROOK
board[0][7] = ROOK
board[7][0] = ROOK
board[7][7] = ROOK

for i in range(8):
    for j in range(8):
        print(board[i][j], end = " | ")
    print("\n")

print("-_____|_____-")

KNIGHT = "^"
board[4][2] = KNIGHT
for i in range(8):
    for j in range(8):
        print(board[i][j], end = " | ")
    print("\n")

print("-_____-")

PAWN = "8"
board[3][4] = PAWN

for i in range(8):
    for j in range(8):
        print(board[i][j], end = " | ")
    print("\n")

print("-_____-")
```

```
========== RESTART: D:/Python content/
ROOK | - | - | - | - | - | - | ROOK |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

ROOK | - | - | - | - | - | - | ROOK |

-_____-
ROOK | - | - | - | - | - | - | ROOK |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | ^ | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

ROOK | - | - | - | - | - | - | ROOK |

-_____-
ROOK | - | - | - | - | - | - | ROOK |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | 8 | - | - | - |

- | - | ^ | - | - | - | - | - |

- | - | - | - | - | - | - | - |

- | - | - | - | - | - | - | - |

ROOK | - | - | - | - | - | - | ROOK |

-_____-
```

Multidimensional nature of lists: advanced applications

To find any element of a two-dimensional list, you have to use two coordinates:

- a vertical one (row number)
- and a horizontal one (column number).

First, you have to decide which data type would be adequate for this application. In this case, a float would be best, since this thermometer is able to measure the temperature with an accuracy of 0.1 ℃.

Then you take an arbitrary decision that the rows will record the readings every hour on the hour (so the row will have 24 elements) and each of the rows will be assigned to one day of the month (let's assume that each month has 31 days, so you need 31 rows). Here's the appropriate pair of comprehensions (h is for hour, d for day):

```
temps = [[0.0 for h in range(24)] for d in range(31)]
#
# The matrix is magically updated here.
#

total = 0.0

for day in temps:
    total += day[11]

average = total / 31

print("Average temperature at noon:", average)
```

Multidimensional nature of lists: advanced applications import random

```
1  temps = [[0.0 for h in range(24)] for d in range(31)]
2  #
3  # The matrix is magically updated here.
4  #
5
6  highest = -100.0
7
8  for day in temps:
9      for temp in day:
10         if temp > highest:
11             highest = temp
12
13 print("The highest temperature was:", highest)
```

Console >_

```
The highest temperature was: 0.0
```

Examples

```
n = 5

a = [[0]*5] *n

for i in range(n):
    print(a[i])

print()
print(len(a))
print()

a[0][2] = 666

for i in range(n):
    print(a[i])
```

```
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]

5

[0, 0, 666, 0, 0]
[0, 0, 666, 0, 0]
[0, 0, 666, 0, 0]
[0, 0, 666, 0, 0]
[0, 0, 666, 0, 0]
>>>
```

```
n = 5

matrix5 = [[0 for j in range(n)] for i in range(n)]
print("Sixe is:", n,"x", n)
print(matrix5)

print()
print(matrix5[0])
print(matrix5[1])
print(matrix5[2])
print(matrix5[3])
print(matrix5[4])
print()

for i in range(n):
    print(matrix5[i])

print()
matrix5[0][0] = 99
for i in range(n):
    print(matrix5[i])
```

```
Sixe is: 5 x 5
[[0, 0, 0, 0, 0], [0, 0, 0
, 0]]

[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]

[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]

[99, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
```

```
# A four-column/four-row table - a two dimensional array (4x4)

table = [[":(", ":)", ":(", ":)"],
         [":)", ":(", ":)", ":)"],
         [":(", ":)", ":)", ":("],
         [":)", ":)", ":)", ":("]]

print(table)
print(table[0][0])  # outputs: ':('
print(table[0][3])  # outputs: ':)'
```

```
# Cube - a three-dimensional array (3x3x3)

cube = [[[':(', 'x', 'x'],
         [':)', 'x', 'x'],
         [':(', 'x', 'x']],

        [[':)', 'x', 'x'],
         [':(', 'x', 'x'],
         [':)', 'x', 'x']],

        [[':(', 'x', 'x'],
         [':)', 'x', 'x'],
         [':)', 'x', 'x']]]

print(cube)
print(cube[0][0][0])  # outputs: ':('
print(cube[2][2][0])  # outputs: ':)'
```

```python
rooms = [[[False for r in range(20)] for f in range(15)] for t in range(3)]
```

Three-dimensional arrays

```python
rooms = [[[0 for r in range(20)] for f in range(15)] for t in range(3)]

rooms[1][9][13] = 1


rooms[0][4][1] = 0

vacancy = 0

for room_number in range(20):
    if not rooms[2][14][room_number]:
        vacancy += 1

print(vacancy)
print(rooms)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
[
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
],
 [
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
]
20
```

Key takeaways

1. List comprehension allows you to create new lists from existing ones in a concise and elegant way. The syntax of a list comprehension looks as follows:

```
[expression for element in list if conditional]
```

which is actually an equivalent of the following code:

```
for element in list:
    if conditional:
        expression
```

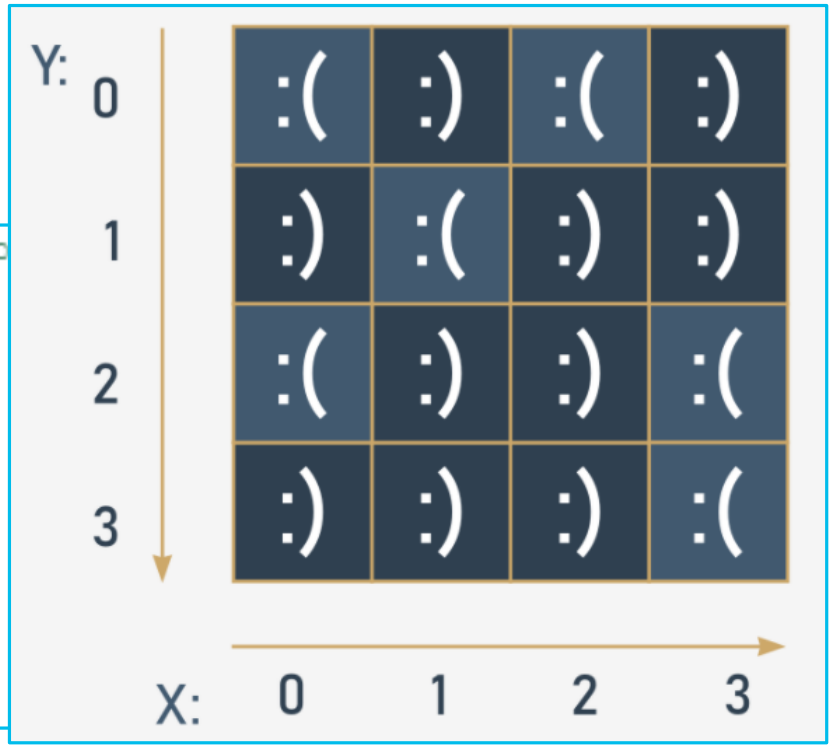
Here's an example of a list comprehension - the code creates a five-element list filled with with the first five natural numbers raised to the power of 3:

```
cubed = [num ** 3 for num in range(5)]
print(cubed)  # outputs: [0, 1, 8, 27, 64]
```

Key takeaways

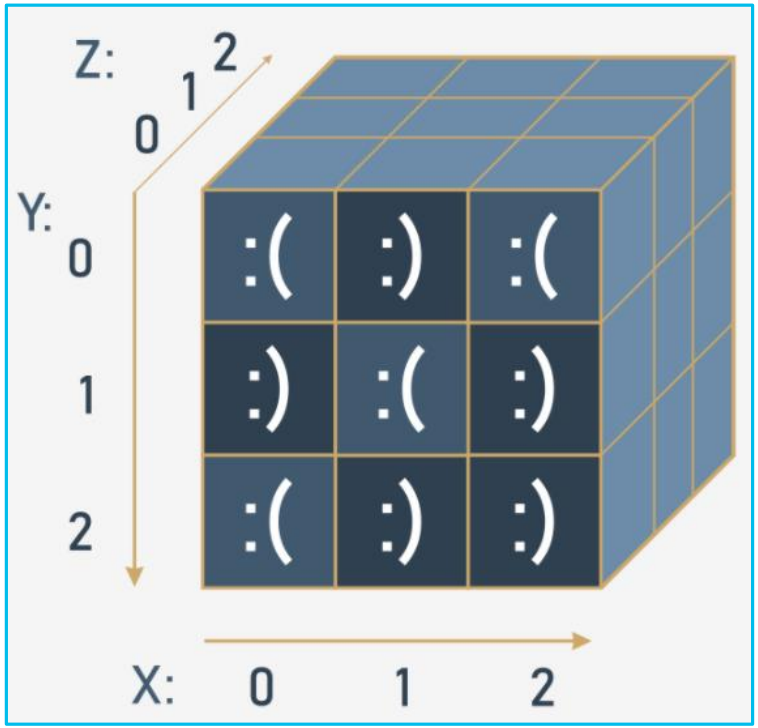
2. You can use nested lists in Python to create matrices (i.e., two-dimensional lists). For example:

```
# A four-column/four-row table - a two dimensio

table = [[":(", ":)", ":(", ":)"],
         [":)", ":(", ":)", ":)"],
         [":(", ":)", ":)", ":("],
         [":)", ":)", ":)", ":("]]

print(table)
print(table[0][0])   # outputs: ':('
print(table[0][3])   # outputs: ':)'
```



```
# Cube - a three-dimensional array (3x3x3)

cube = [[[':(', 'x', 'x'],
         [':)', 'x', 'x'],
         [':(', 'x', 'x']],

        [[':)', 'x', 'x'],
         [':(', 'x', 'x'],
         [':)', 'x', 'x']],

        [[':(', 'x', 'x'],
         [':)', 'x', 'x'],
         [':)', 'x', 'x']]]

print(cube)
print(cube[0][0][0])  # outputs: ':('
print(cube[2][2][0])  # outputs: ':)'
```

3. You can nest as many lists in lists as you want, and therefore create n-dimensional lists, e.g., three-, four- or even sixty-four-dimensional arrays. For example:



Examples

```
students = ['Ivan', 'Masha', 'Sasha']
students += ['Olga']
students += 'Olga'
print(students)
```

```
>>> s = ['d', 'dd']
>>> s += [1]
>>> s
['d', 'dd', 1]
>>> s += 'sd'
>>> s
['d', 'dd', 1, 's', 'd']
>>> s += 'oleg'
>>> s
['d', 'dd', 1, 's', 'd', 'o', 'l', 'e', 'g']
>>> s += 2999
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    s += 2999
TypeError: 'int' object is not iterable
>>> s
['d', 'dd', 1, 's', 'd', 'o', 'l', 'e', 'g']
>>>
```

Home work 3.5 split()

```
put string of numbers by space: 4 -55 888 -212
Your input: 4 -55 888 -212
Your input after split: ['4', '-55', '888', '-212']
after append in list: [4, -55, 888, -212]
Sum(list):
625
>>>
```


Write a program that accepts one line(string) of integers as input. The program should output the sum of these numbers.

Use **s** variable:

- s=input()
- **li1 = s**.split(" ")
- numbers = []
- For I in ….
  - numbers.append(int(li[i]))
- sum(numbers)
- print(s, numbers)

INVESTIGATE:

- https://www.w3schools.com/python/python_ref_string.asp

```
>>> a1
['1', '2', '3', '4', '5']
>>> a1[0]
'1'
>>> int(a1[0])
1
>>>
```

```
>>> s = '1 2 3 4 5'
>>> li = s.split(' ')
>>> li
['1', '2', '3', '4', '5']
```

Home work 3.5 split()

```
>>> a = input()
1,2,3,4,5
>>> a
'1,2,3,4,5'
>>> a1 = a.split(',')
>>> a1
['1', '2', '3', '4', '5']
>>>
```

```
>>> s = '1 2 3 4 5'
>>> li = s.split(' ')
>>> li
['1', '2', '3', '4', '5']
>>> liInt = []
>>> for i in li:
```

```
>>> rs = []
>>> a
'1,2,3,4,5'
>>> a1
['1', '2', '3', '4', '5']
>>> resInt = []
>>> for i in range(len(  )):
```

```
>>> resInt
[1, 2, 3, 4, 5]
>>> sum(resInt)
15
>>>
```

Home work 3.6
avg

-5 12
i in range(-5, 12+1)
if i % 3 == 0:
li.append(i)

or use list gen

```
>>> ls = [ for  if  ]
>>> ls
[-3, 0, 3, 6, 9, 12]
>>> sum( )/len( )
4.5
```

```
>>>
==========
-3
25
10.5
>>>
==========
-5
12
4.5
>>>
==========
1
33
18.0
```

Write a program that reads two numbers aa and bb from the keyboard, calculates and displays the arithmetic mean of all numbers from the segment [a; b], **which are multiples of 3**.

In the example below, the arithmetic mean is calculated for numbers on the segment [-5; 12]. The total number of numbers **divisible by 3** on this segment is **6**: -3, 0, 3, 6, 9, 12. Their arithmetic mean is **4.5**

The program receives input to the intervals, within which there is always at least one number that is divisible by 3.

Use:

- float() input()
- print()
- while/for
- %
- and all that you want

ЗАДАНИЯ

1) Прорешать всю классную работу
2) Выполнить все домашние задания

Почитать:

1) Byte of Python –
Прочитать страницы -
стр. 47-54
стр. 55-63 стр. 85-87

Крайний срок сдачи 03/10 в 21:00 (можно раньше, но не позже)

ЗАДАНИЯ

Название файлов, которые вы отправляете мне в telegram:

Vasia_Pupkin_class_work_L4_0.py

+все задания ОДНИМ ФАЙЛОМ - Vasia_Pupkin_L4_0.py

Формат сообщения которое вы присылаете мне

(после полного выполнения домашнего задания, только один раз) в Telegram:

Добрый день/вечер. Я Вася Пупкин, и это мои домашние задания к лекции 4 часть 0.

И отправляете файлы

Крайний срок сдачи 03/10 в 21:00 (можно раньше, но не позже)

https://docs.github.com/articles/using-pull-requests

Create your possibilities. Bye bye.