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:

- **the Boolean data type;**
- **relational operators;**
- **making decisions in Python (if, if-else, if-elif, else)**
- **how to repeat code execution using loops (while, for)**
- **how to perform logic and bitwise operations in Python;**
- lists in Python (constructing, indexing, and slicing; content manipulation)
- how to sort a list using bubble-sort algorithms;
- multidimensional lists and their applications.

Python supports the usual logical conditions from mathematics:

b = 0

a = 2

b += 1

- Equals: a **==** b
- Not Equals: a **!=** b
- Less than: a **<** b
- Less than or equal to: a **<=** b
- Greater than: a **>** b
- Greater than or equal to: a **>=** b

Priority	Operator	
1	`+`, `-`	unary
2	`**`	
3	`*`, `/`, `//`, `%`	
4	`+`, `-`	binary
5	`<`, `<=`, `>`, `>=`	
6	`==`, `!=`	

Example

```python
n = int(input("Enter a number: "))

print(n > 100)
print(n < 100)


#use it all
print(n == 55)
print(n != 55)

print(n >= 100)
print(n <= 100)
```

output:

```
Console >_

Enter a number: 55
False
True
True
False
False
True
```

Example
x == 22332
“sfd” == “sf”

Try to use:

x = “10.”

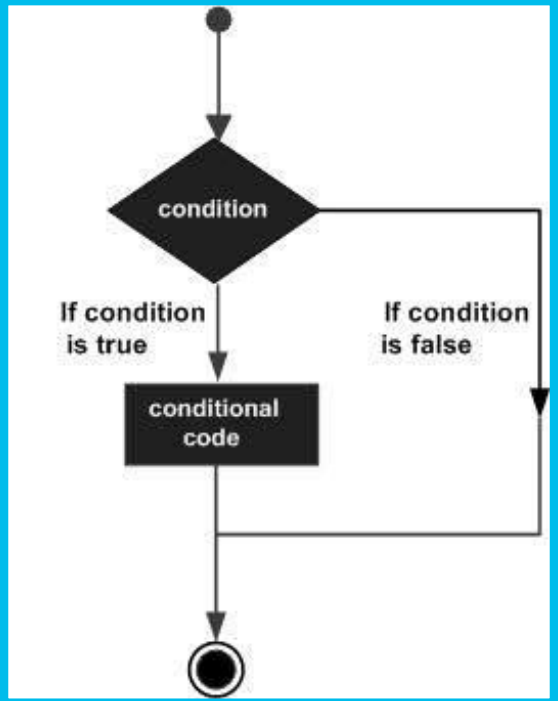
print(x, type(x))


```
1  x = 10
2  y = 20
3  z = 20
4  print (x == y) #output: False
5  print (z == y) #output: True
6  print (x != y) #output: True
7  print (x > 20) #output: False
8  print (x < y) #output: True
9  print(x >= 10) #output: True
10 print (y <= 20) #output: True
```

output:

```
False
True
True
False
True
True
True
```

Conditions and conditional execution



- Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.
- Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome.
- You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.

This conditional statement consists of the following, strictly necessary, elements in this and this order only:

```
if true_or_not:
    do_this_if_true
```

- the **if** keyword;
- one or more white spaces;
- an **expression** (a question or an answer) whose value will be interpreted solely in terms of **True** (when its value is non-zero) and **False** (when it is equal to zero);
- a **colon** followed by a newline;
- an **indented instruction** or **set of instructions** (at least one instruction is absolutely required);
- the indentation may be achieved in two ways - by inserting a particular number of spaces (the recommendation is to **use four spaces of indentation**), or by using the **tab** character; **note**: if there is more than one instruction in the indented part, the indentation should be the same in all lines; even though it may look the same if you use tabs mixed with spaces, it's important to make all indentations exactly the same - Python 3 does not allow mixing spaces and tabs for indentation.

Knowing what conditions influence our behavior, and assuming that we have the parameterless functions go_for_a_walk() and have_lunch(), we can write the following snippet:

```
if the_weather_is_good:
    go_for_a_walk()
have_lunch()
```

```
if sheep_counter >= 120: # Evaluate a test expression
    sleep_and_dream() # Execute if test expression is True
```

Conditional execution: the **if statement**

```
if sheep_counter >= 120:
    make_a_bed()
    take_a_shower()
    sleep_and_dream()
feed_the_sheepdogs()
```

Conditional execution: the **if-else statement**

The **if-else** execution goes as follows:

- if the condition evaluates to True (its value is not equal to zero), the perform_if_condition_true statement is executed, and the conditional statement comes to an end;
- if the condition evaluates to False (it is equal to zero), the perform_if_condition_false statement is executed, and the conditional statement comes to an end.

```
if true_or_false_condition:
    perform_if_condition_true
else:
    perform_if_condition_false
```

The **if-else** statement: more conditional execution

del ":"

If statement, without indentation (will raise an error):

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

```
  File "main.py", line 4
    print("b is greater than a") # you will get an error
        ^
IndentationError: expected an indented block
```

```
if the_weather_is_good:
    go_for_a_walk()
else:
    go_to_a_theater()
have_lunch()
```

```
if the_weather_is_good:
    go_for_a_walk()
    have_fun()
else:
    go_to_a_theater()
    enjoy_the_movie()
have_lunch()
```

Try to use:

print(type(amount), type(discount))

Nested if-else statements

```
1  amount = int(input("Enter amount: "))
2
3  if amount<1000:
4      discount = amount*0.05
5      print ("Discount",discount)
6  else:
7      discount = amount*0.10
8      print ("Discount",discount)
9
10 print ("Net payable:",amount-discount)
```

```
Enter amount: 20000
Discount 2000.0
Net payable: 18000.0
```


Here are two important points:

- this use of the if statement is known as **nesting**; remember that **every else refers to the if which lies at the same indentation level**; **you need to know this** to determine how the ifs and elses pair up;
- consider **how the indentation improves readability**, and makes the code **easier to understand and trace**.

```
if the_weather_is_good:
    if nice_restaurant_is_found:
        have_lunch()
    else:
        eat_a_sandwich()
else:
    if tickets_are_available:
        go_to_the_theater()
    else:
        go_shopping()
```

```
1  amount = int(input("Enter amount: "))
2
3  if amount<1000:
4      discount = amount*0.05
5      print ("Discount",discount)
6  elif amount<5000:
7      discount = amount*0.10
8      print ("Discount",discount)
9  else:
10     discount = amount*0.15
11     print ("Discount",discount)
12
13 print ("Net payable:",amount-discount)
```

```
Enter amount: 900
Discount 45.0
Net payable: 855.0
```


Some additional attention has to be paid in this case:

- you mustn't use else without a preceding if;
- **else is always the last branch of the cascade**, regardless of whether you've used elif or not;
- **else is an optional part of the cascade, and may be omitted**;
- if there is an **else** branch in the cascade, **only one of all the branches** is executed;
- if there is no else branch, it's possible that none of the available branches is executed.

```
if the_weather_is_good:
    go_for_a_walk()
elif tickets_are_available:
    go_to_the_theater()
elif table_is_available:
    go_for_lunch()
else:
    play_chess_at_home()
```

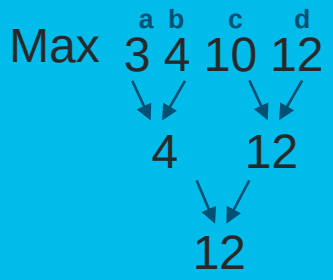
Example

```python
# Read two numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

# Choose the larger number
if number1 > number2:
    larger_number = number1
else:
    larger_number = number2

# Print the result
print("The larger number is:", larger_number)
```

Console >_

```
Enter the first number: 44
Enter the second number: 33
The larger number is: 44
```

Example



Home work 2.1

It's time to complicate the code - let's **find the largest of four numbers**.

- input()
- int() or float()
- < > == <= >=
- if/elif/else
- **type()**
- **comments!!!#####**

Good luck(^_^)

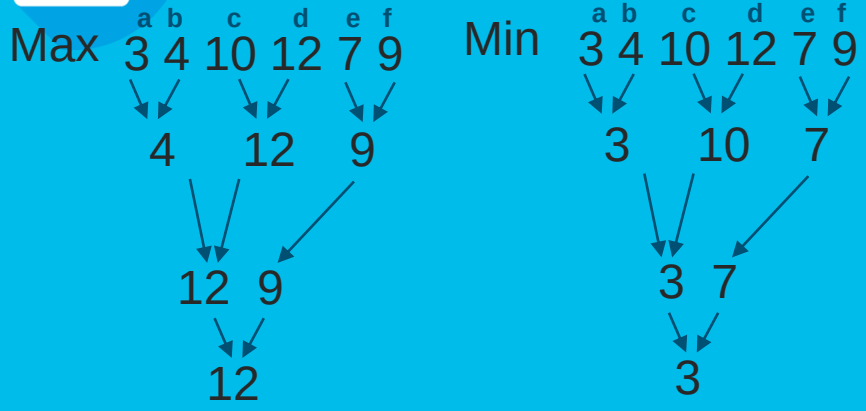
Pseudocode and introduction to loops

```
1 largest_number = -999999999
2 number = int(input())
3 if number == -1:
4     print(largest_number)
5     exit()
6 if number > largest_number:
7     largest_number = number
8 # Go to line 02
```

Note.

Python often comes with a lot of built-in functions that will do the work for you. For example, to find the largest number of all, you can use a Python built-in function called max(), min(). You can use it with multiple arguments.

Example



Home work 2.2

It's time to complicate the code - let's find the max and min of 6 numbers.

- **input()**
- **int() or float()**
- ***if elif else**
- **max(), min()**
- type()
- **comments!!! #####**

Test Data

Sample input: `spathiphyllum`

Expected output: `No, I want a big Spathiphyllum!`

Sample input: `pelargonium`

Expected output: `Spathiphyllum! Not pelargonium!`

Sample input: `Spathiphyllum`

Expected output: `Yes - Spathiphyllum is the best plant ever!`

Home work 2.3

Spathiphyllum, more commonly known as a peace **lily** or white sail plant, is one of the most popular indoor houseplants that filters out harmful toxins from the air. Some of the toxins that it neutralizes include benzene, formaldehyde, and ammonia.

Imagine that your computer program loves these plants. Whenever it receives an input in the form of the word Spathiphyllum, it involuntarily shouts to the console the following string: "Spathiphyllum is the best plant ever!"

Write a program that utilizes the concept of conditional execution, takes a string as input, and:

- prints the sentence "Yes - Spathiphyllum is the best plant ever!" to the screen if the inputted string is "Spathiphyllum" (upper-case)
- prints "No, I want a big Spathiphyllum!" if the inputted string is "spathiphyllum" (lower-case)
- prints "Spathiphyllum! Not [input]!" otherwise. Note: [input] is the string taken as input.

Test your code using the data we've provided for you. And get yourself a Spathiphyllum, too!

Home work 2.4
comment

Test Data

Sample input: `2000`

Expected output: `Leap year`

Sample input: `2015`

Expected output: `Common year`

Sample input: `1999`

Expected output: `Common year`

Sample input: `1996`

Expected output: `Leap year`

Sample input: `1580`

Expected output: `Not within the Gregorian calendar period`

Since the introduction of the Gregorian calendar (in 1582), the following rule is used to determine the kind of year:

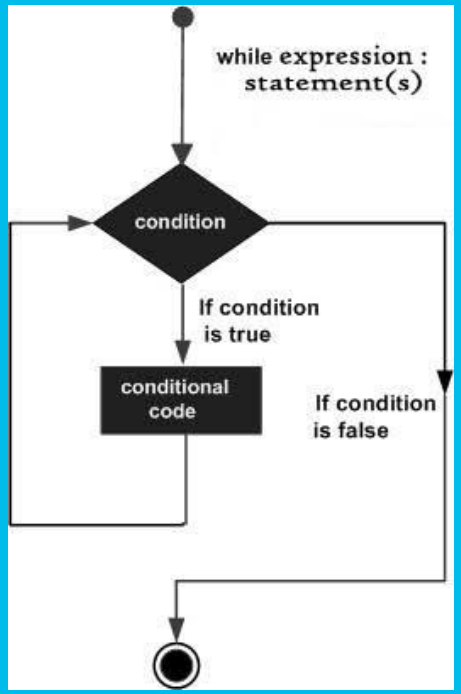
- **if the year number isn't divisible by four, it's a common year;**
- **otherwise, if the year number isn't divisible by 100, it's a leap year;**
- **otherwise, if the year number isn't divisible by 400, it's a common year;**
- **otherwise, it's a leap year.**

Look at the code in the editor - it only reads a year number, and needs to be completed with the instructions implementing the test we've just described.

The code should output one of two possible messages, which are Leap year or Common year, depending on the value entered.

It would be good to verify if the entered year falls into the Gregorian era, and output a warning otherwise: Not within the Gregorian calendar period. Tip: use the != and % operators.

Looping your code with **while**




```
while conditional_expression:
    instruction
```

```
while conditional_expression:
    instruction_one
    instruction_two
    instruction_three
    :
    :
    instruction_n
```

An infinite loop

```
while True:
    print("I'm stuck inside a loop.")
```

to stop it use
Ctr+Z or
Ctrl+C

```
1  count = 0
2  while (count < 9):
3      print ('The count is:', count)
4      count = count + 1
5
6  print ("Good bye!")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

```
1  # Store the current largest number here.
2  largest_number = -999999999
3
4  # Input the first value.
5  number = int(input("Enter a number or type -1 to stop: "))
6
7  # If the number is not equal to -1, continue.
8  while number != -1:
9      # Is number larger than largest_number?
10     if number > largest_number:
11         # Yes, update largest_number.
12         largest_number = number
13     # Input the next number.
14     number = int(input("Enter a number or type -1 to stop: "))
15
16 # Print the largest number.
17 print("The largest number is:", largest_number)
```

The **while** loop: r examples

```
# A program that reads a sequence of numbers
# and counts how many numbers are even and how many are odd.
# The program terminates when zero is entered.

odd_numbers = 0
even_numbers = 0

# Read the first number.
number = int(input("Enter a number or type 0 to stop: "))

# 0 terminates execution.
while number != 0:
    # Check if the number is odd.
    if number % 2 == 1:
        # Increase the odd_numbers counter.
        odd_numbers += 1
    else:
        # Increase the even_numbers counter.
        even_numbers += 1
    # Read the next number.
    number = int(input("Enter a number or type 0 to stop: "))

# Print results.
print("Odd numbers count:", odd_numbers)
print("Even numbers count:", even_numbers)
```

bar += 1

bar = bar +1

The **while** loop: more examples

```
1 counter = 5
2 while counter != 0:
3     print("Inside the loop.", counter)
4     counter -= 1
5 print("Outside the loop.", counter)
6
```

Looping your code with **for**
0-99

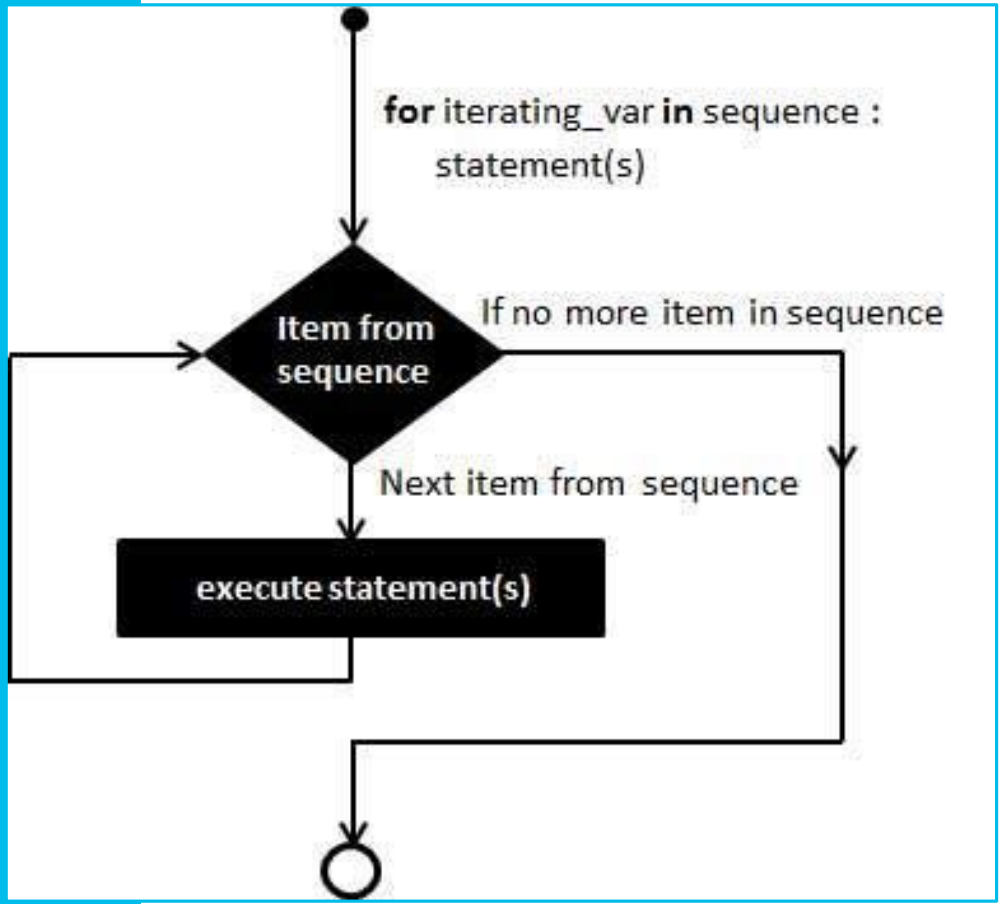
```
for i in range(100):
    # do_something()
    pass
```


There are some new elements. Let us tell you about them:

- the **for** keyword opens the for loop; note - there's no condition after it; you don't have to think about conditions, as they're checked internally, without any intervention;
- any variable after the for keyword is the control variable of the loop; it counts the loop's turns, and does it automatically;
- the **in** keyword introduces a syntax element describing the range of possible values being assigned to the control variable;
- the **range()** function (this is a very special function) is responsible for generating all the desired values of the control variable; in our example, the function will create (we can even say that it will feed the loop with) subsequent values from the following set: **0, 1, 2 .. 97, 98, 99**; note: in this case, the range() function **starts its job from 0** and finishes it one step (one integer number) before the value of its argument;

note the **pass** keyword inside the loop body - it does nothing at all; it's an empty instruction - we put it here because the for loop's syntax demands at least one instruction inside the body (by the way - if, elif, else and while express the same thing)

Looping your code with **for**



```
for i in range(10):
    print("The value of i is currently", i)
```

Looping your code with **for**

```
for i in range(2, 8):
    print("The value of i is currently", i)
```

Looping your code with **for**

```
power = 1
for expo in range(16):
    print("2 to the power of", expo, "is", power)
    power *= 2
```

- Example:

```
for var in list(range(5)):
    print (var)
```

Output:

```
0
1
2
3
4
```

- Example:

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

Output:

```
2 3 4 5
```

- Example:

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

- Output:

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

Home work 2.5 Count mississippily.

The idea behind it is that adding the word Mississippi to a number when counting seconds aloud makes them sound closer to clock-time, and therefore "**one Mississippi**, **two Mississippi**, **three Mississippi**" will take approximately an actual three seconds of time! It's often used by children playing hide-and-seek to make sure the seeker does an honest count.

Your task is very simple here: write a program that uses a for loop to "count mississippily" to five. Having counted to five, the program should print to the screen the final message "**Ready or not, here I come!**"

```
1  import time
2
3  # Write a for loop that counts to five.
4      # Body of the loop - print the loop iteration number and the word "Mississippi".
5      # Body of the loop - use: time.sleep(1)
6
7  # Write a print function with the final message.
8
```

The **break** and **continue** statements

```
# break - example

print("The break instruction:")
for i in range(1, 6):
    if i == 3:
        break
    print("Inside the loop.", i)
print("Outside the loop.")


# continue - example

print("\nThe continue instruction:")
for i in range(1, 6):
    if i == 3:
        continue
    print("Inside the loop.", i)
print("Outside the loop.")
```

Console >_

```
The break instruction:
Inside the loop. 1
Inside the loop. 2
Outside the loop.

The continue instruction:
Inside the loop. 1
Inside the loop. 2
Inside the loop. 4
Inside the loop. 5
Outside the loop.
```

The bre
continu

```
largest_number = -99999999
counter = 0

while True:
    number = int(input("Enter a number or type -1 to end program: "))
    if number == -1:
        break
    counter += 1
    if number > largest_number:
        largest_number = number

if counter != 0:
    print("The largest number is", largest_number)
else:
    print("You haven't entered any number.")
```

Console >_

```
Enter a number or type -1 to end program: 1
Enter a number or type -1 to end program: 2
Enter a number or type -1 to end program: 3
Enter a number or type -1 to end program: -1
The largest number is 3
```

The **break** and **continue** statements

```
largest_number = -99999999
counter = 0

number = int(input("Enter a number or type -1 to end program: "))

while number != -1:
    if number == -1:
        continue
    counter += 1

    if number > largest_number:
        largest_number = number
    number = int(input("Enter a number or type -1 to end program: "))

if counter:
    print("The largest number is", largest_number)
else:
    print("You haven't entered any number.")
```

Console >_

```
Enter a number or type -1 to end program: 1
Enter a number or type -1 to end program: 2
Enter a number or type -1 to end program: 3
Enter a number or type -1 to end program: -1
The largest number is 3
```

Home work 2.6

```
name = input()
name = name.upper()
for i in name:
        if i in “AEIOU”:
                continue
        print(i)
```

Test data

Sample input: `Gregory`

Expected output:

```
G
R
G
R
Y
```

The continue statement is used to skip the current block and move ahead to the next iteration, without executing the statements inside the loop.

It can be used with both the while and for loops.

Your task here is very special: you must design a vowel eater! Write a program that uses:

- a for loop;
- the concept of conditional execution (if-elif-else)
- the continue statement.

Your program must:

- ask the user to enter a word **input()**;
- use **user_word = user_word.upper()** to convert the word entered by the user to upper case; we'll talk about the so-called string methods and the **upper()** method very soon - don't worry;
- use conditional execution and **the continue statement to "eat"** the following vowels **A, E, I, O, U** from the inputted word;

print the uneaten letters to the screen, each one of them on a separate line.

Test your program with the data we've provided for you.

The **while** loop and the **else** branch

```
numbers = [11,33,55,39,55,75,37,21,23,41,13]

for num in numbers:
    if num%2 == 0:
        print ('the list contains an even number')
        break
else:
    print ('the list does not contain even number')
```

```
the list does not contain even number
```

```
count = 0
while count < 5:
    print (count, " is  less than 5")
    count = count + 1
else:
    print (count, " is not less than 5")
```

```
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

The loop's else branch is always executed once, regardless of whether the loop has entered its body or not.

```
i = 1
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

Console >_

```
1
2
3
4
else: 5
```

Computer logic

and or not

Try to use:

A = True

B = False

print(A and B, type(A))

One logical conjunction operator in Python is the word and. It's a binary operator with a priority that is lower than the one expressed by the comparison operators.

Computer logic **and**

Argument `A`	Argument `B`	`A and B`
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

```
a = True
print(a, type(a)) #<class 'bool'>
print(not a) #False
```

Computer logic **and**

```
1  i = 0
2  x = 0
3  while (i < 10 and x < 5):
4      x = i
5      print(i, end= " ")
6      i += 1
```

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

One logical conjunction operator in Python is the word and. It's a binary operator with a priority that is lower than the one expressed by the comparison operators.

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

```
Both conditions are True
```

Try to use:

A = True

B = False

print(A or B, type(A))

Computer logic

or

A disjunction operator is the word or. It's a binary operator with a lower priority than and (just like + compared to *). Its truth table is as follows:

Argument A	Argument B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

Computer logic
or

A disjunction operator is the word or. It's a binary operator with a lower priority than and (just like + compared to *). Its truth table is as follows:

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

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

```
1  i = 0
2  x = 0
3  while (i < 10 or x < 5):
4      x = i
5      print(i, end= " ")
6      i += 1
```

```
0 1 2 3 4 5 6 7 8 9
```

ЗАДАНИЯ

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

Почитать:

1) Byte of Python

Прочитать страницы -

стр. 47-54

стр. 55-63

Крайний срок сдачи 28/09 в 21:00 (можно раньше, но не позже)

ЗАДАНИЯ

Название файлов, которые вы отправляете мне в telegram:

Vasia_Pupkin_class_work_L2_3.py

+все задания ОДНИМ ФАЙЛОМ - Vasia_Pupkin_L2_3.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 2 часть 3.

И отправляете файл/-лы

Крайний срок сдачи 28/09 в 21:00 (можно раньше, но не позже)

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

Create your possibilities. Bye bye.