

Python Basics



Why Python?

- It's a popular programming language.
- It's got many applications (automation, web development, data science).
- Its syntax is very similar to written English.
- It's straightforward to learn.
- Python files or scripts have the `.py` extension

PYTHON

```
print("Hello World")
```

VS.

JAVA

```
1 public class HelloWorld {  
2  
3     public static void main(String[] args)  
4     {  
5         System.out.println("Hello world");  
6     }  
7 }
```

Data Types

Basic Data Types

- Booleans (True/False)
- Integers (Whole numbers)
- Floats (Decimal Numbers)
- Strings (Sequence of characters “words”)

Complex Data Types

- Lists (Arrays)
- Tuples (Immutable Arrays)
- Dictionaries (Hashed Maps)

Variables

- “Boxes” that store data to be used later in the program.
- Python is dynamically typed.
- To declare a variable just write: *var_name = value*
- All variable names must start with either a lowercase or an uppercase letter.
- Use variable names that are related to the data you store on them.

Conventions

- Use lowercase letters for variables whose values are meant to change.
- Use uppercase letters for constants (They don't exist at all in Python).
- Use snake casing (*to_write_like_this*) for variables whose names have 2 or more words.

Comments

- Text ignored by the Python interpreter used to leave notes in your code
- Programmers use them to let others know how their code works, what needs to get done, or for any other thing.
- Sometimes the other programmer will be you after some weeks.
- ALWAYS COMMENT YOUR CODE IF YOU'RE GOING TO FORGET HOW IT WORKS IN THE NEXT 2 WEEKS
- Comments in Python are preceded by the # character

```
# I am the coolest Python comment that's ever existed and that'll ever exist.|
```

Command Line I/O Operations

STDIN (Standard Input)

- It's input data that comes from the keyboard.
- Use the *input()* function to get STDIN from the user.
- Save the result in a variable to be able to use it later.
- Ex: `name = input("What is your name? ")`

STDOUT (Standard Output)

- A program's output printed to the console.
- Use the *print()* function to display the results of your program to the console.
- Ex: `print("I am the best programmer that has ever existed.")`

Small Exercise 1

Write a short program that asks the user his name, his intended major and print it to the console.

Arithmetic Operators

- Addition (+): `2 + 3`
- Subtraction (-): `4 - 2`
- Multiplication (*): `6 * 8`
- Division (/): `16 / 2`
- Power (**): `2 ** 4`
- Modulo (%): `15 % 2`
- THE MODULO OPERATOR IS USED A LOT TO CHECK FOR EVEN AND ODD NUMBERS.

Type Conversion

- Use the *bool()* function to convert an object to its boolean equivalent
- Use the *int()* function to convert numeric data to an integer.
- Use the *float()* function to convert numeric data to a float.
- Use the *str()* function to convert data to string.

Why is this important?

- The *input()* function returns all data as strings.
- You can't concatenate numbers with strings to print them (We'll see this later).

Small Exercise 2

Make a small program that gets two numbers from the user and executes all arithmetic operations covered with those numbers. Print each result to the console.

Strings

- Immutable data types
- Concatenation: Joining two strings together.
- Ex: `"I am" + " the best"`
- Indexing: Referring to characters of a string with numbers
- THE FIRST CHARACTER IS ALWAYS REFERENCED WITH INDEX 0
- Ex: `"I am the best"[0]`
- Slicing: Extracting pieces of strings.
- First index is inclusive. Last index is exclusive.
- Ex: `"I am the best"[2:4]`
- Use the `len()` function to get how long a string is.
- Ex: `len("I am the best")`
- There are many more ways to deal with strings called **STRING METHODS**. Find more info about them here:
- https://www.w3schools.com/python/python_ref_string.asp

Small Exercise 3

Take input from STDIN, get it's length and make a substring that takes everything but the last character.

Comparison Operators

- Use them to compare/contrast your data.
 - Equality: `4 == 4`
 - Inequality: `"Cool" != "cool"`
 - Greater than: `5 > 6`
 - Lesser than: `3 < 4`
 - *Greater or equal*: `100 >= 99`
 - *Smaller or equal to*: `60 <= 80`
-
- More info and examples may be found here:
 - https://www.w3schools.com/python/gloss_python_comparison_operators.asp

Logical Operators


- They are used to construct complex logical statements
- AND: True when all expressions are true
- Ex: `4 == 4 and 5 > 6`
- OR: True when at least one expression is true
- Ex: `4 == 4 or 5 > 6`
- NOT: Inverse the “veracity” of a logical statement
- Ex: `not 15 < 10`

in keyword

- Check if a value is present in a string, list, tuple, etc.
- Ex: `"galaxy" in "A long time ago in a galaxy far, far away"`
- *More info on logical operators may be found here:*
- *https://www.w3schools.com/python/gloss_python_logical_operators.asp*

If-statement

- Used to make choices according to a condition.
- Allows your code to follow your logic.



```
name = 'Jason'
if name == 'Jason':
    print("Hello Jason, Welcome")
else:
    print("Sorry, I don't know you")
```

Elif

For cases that involve more than two possibilities, use the elif statement which stands for (else if).

```
litter = int(input("How many puppies were born?"))

if litter <= 5:
    print("good size")

elif litter == 6:
    print("Just right")

elif litter == 7:
    print("large litter")

else:
    print("goodnes me")
```


Small Exercise 4

Write a small program that asks the user how they feel today.

- If they say “bad” or “Bad” print out: “How can I help”?
- If they say “regular” or “Regular” print out: “You had a boring day.”
- If they say “good” or “Good” print out: “Nice”.
- If they say “awesome” or “Awesome” print out: “You conquered the world today.”
- For all other cases print out: “Meh, I didn’t even care in the first place.”

While Loop

- In programming, while loops are used to execute tasks repeatedly until a certain condition tells them to stop.
- Use them to execute code as long as a certain condition is true.

```
answer = "N"
while answer != "Y":
    print("Are we there yet?")
    answer = input("Please respond Y or N")

print("Yay! We are there!")
```

WARNING

- While loops WILL run FOREVER unless you make a condition that WILL cut them off.
- You must manually cancel them by exiting the program or with the shortcut *Ctrl + C*
- NEVER WRITE INFINITE LOOPS !!!

```
while True:  
    print("I am an infinite Loop.")  
    print("I've got eternal life and you don't.")  
    print("Ha ha ha ha ha.")
```

Small Exercise 5

Take the program from Small Exercise 4 to keep asking the user how they feel until they enter the string “Stop”

For Loops

- They are used to iterate through a sequence of objects.
- They are used a lot to address individually items in strings, lists, and ranges of numbers.

```
7
8
9
10     for i in range(1, 10):
11         | print(i)
12
13
14
```

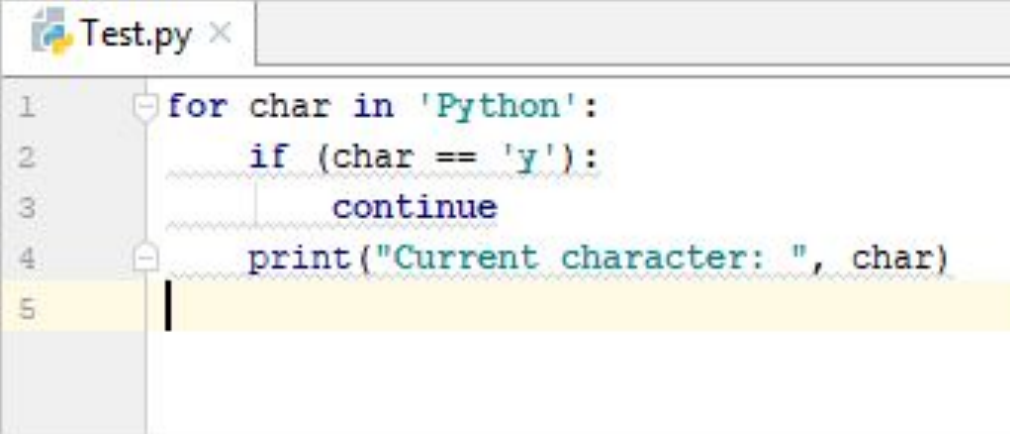
Small Exercise 6

Make a program that prints out the characters of a string from STDIN one at a time.

Special Loop Statements

continue

- Use it to skip an iteration of the loop.



The screenshot shows a code editor window titled 'Test.py'. It contains a Python script with a for loop that iterates over the string 'Python'. Inside the loop, there is an if statement that checks if the current character is 'y'. If it is, the 'continue' statement is used to skip the rest of the loop body and move to the next iteration. Otherwise, the program prints the current character. The code is as follows:

```
1 for char in 'Python':  
2     if (char == 'y'):  
3         continue  
4     print("Current character: ", char)  
5
```

Small Exercise 7

Make a program that prints out all even numbers from 1 to 100.

Special Loop Statements

break

- Use it to force exit from a loop.

```
for i in range(10):  
    print(i)  
    if i == 2:  
        break
```

Small Exercise 8

Make a program that continuously asks the user for a number, but force exit from it when the user does not enter digits.

Functions

- Separate pieces of code used to execute determined tasks.
- They may take input through arguments and may return output.
- You MUST CALL functions in order to execute their code.

```
def are_you_awesome(name):  
    if len(name) > 0:  
        return True  
    else:  
        return False  
  
jose_awesome = are_you_awesome("Jose")  
print(jose_awesome)
```

```
def hello_world():  
    print("Hello, World")
```

```
hello_world()
```

```
def give_something():  
    return "Some value IDK"
```

```
print(give_something())
```

Scope

- Variables outside functions are called *global*. They may be read by the entire program. But not necessarily changed.
- Variables inside a function are called *local*. They cannot be accessed outside the function.
- Have this in mind as you learn how to manage more complex programs that involve the usage of different functions.
- It is recommended to have your code split into different functions that each execute a specific task and then call them inside a *main()* function

Scopes Example

```
grade = 50

print("Global grade is: " + str(grade))

def read_grade():
    grade = 75
    print("Local grade is: " + str(grade))

read_grade()
print("Global grade is: " + str(grade))

def change_grade():
    global grade
    grade = 75
    print("Local grade is: " + str(grade))

change_grade()
print("Global grade is: " + str(grade))
```

Scopes Example with *main()*

```
grade = 50

def main():

    print("Global grade is: " + str(grade))

    read_grade()
    print("Global grade is: " + str(grade))

    change_grade()
    print("Global grade is: " + str(grade))

def read_grade():
    grade = 75
    print("Local grade is: " + str(grade))

def change_grade():
    global grade
    grade = 75
    print("Local grade is: " + str(grade))

# Do not execute this code if the program is imported by another Python File
if __name__ == "__main__":
    main()
```

Docstrings

- The “comments” of functions used to summarize what functions do, what type of data they use as input and what data type they return.
- It's the description of functions that appear when the *help()* function is used.
- It's recommended you add docstrings to your functions unless their names completely describe them.

```
def may_drink(age):  
    """Return True if age is 21 or more. Otherwise return False."""  
    if age >= 21:  
        return True  
  
    else:  
        return False
```

Small Exercise 9

Write two functions, one that returns true when a number is odd and false when a number is even.

Then make a program that continuously asks the user for a number and print out if that number is even or odd.

Exit the program when the user enters “Stop”.

Use comments and docstrings in your code.

Encapsulate the meat of the program (The part of the program that defines control flow and that calls the other functions) inside a *main()* function.

Lists

- Used to store multiple items in one variable
- `subjects = ["math", "science", "history"]`
- In Python you may store objects of any data type inside the same list.
- Use indexing to refer to individual items of a list
- `subjects[0]`
- You may change list elements of a specific index.
- `subjects[0] = "music"`
- Use slicing to extract a sublist from a list
- `[1,2,3,4,5,6][2:4]`
- More ways to deal with lists may be found here:
- https://www.w3schools.com/python/python_ref_list.asp

Small Exercise 10

Make a program that continually asks the user for integers. Then add only even numbers to a list and print that list.

Tuples

- Same as lists but you cannot change the elements it contains.
- You cannot add, replace, or remove elements from a tuple.

Why should they be used?

- Sometimes, you need to with data that does not need to be changed or must not be changed.
- They are a bit quicker to compute than lists.
- Use the *list()* method to convert a tuple to a list
- Use the *tuple()* method to convert a list to a tuple
- `a_tuple = ("Once", "I", "am", "initialized", "you", "cannot", "change", "me")`

Small Exercise 11

Get how many times the numbers 50 appears in the tuple

```
nums = (30, 50, 20, 10, 50, 30, 40, 40, 20, 50, 80, 100, 50)
```

Dictionaries

- Kind of like lists, but instead of using indexes they refer to data stored in them with keys.
- They use key value pairs
- `my_dict = {"person": "I", "is_the_best": True}`
- `my_dict["person"] = "I"`
- Special things you may do with dictionaries may be found here:
- https://www.w3schools.com/python/python_ref_dictionary.asp

Final Exercise

Make a program that takes a string from STDIN, get the letters present in that string and count how many times they repeat.