

PYTHON TUTORIAL FOR BEGINNERS

```
class Python:
```

```
#
```

```
def programin_with_mosh:
```

```
#here my code..
```



```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

Transcribed by: Janner Pareja

Author: Mosh Hamedani

Date: 16 sep. 2020

▼ TABLE OF CONTENT

#

- 1. Introduction**
- 2. What You Can Do With Python**
- 3. My First Python Program**
- 4. Variables**
- 5. Receiving Input**
- 6. Type Conversion**
- 7. Strings**
- 8. Arithmetic Operators**
- 9. Operator Precedence**
- 10. Comparison Operators**
- 11. Logical Operators**
- 12. If Statements**
- 13. Exercise**
- 14. While Loops**
- 15. Lists**
- 16. List Methods**
- 17. For Loops**
- 18. The range() Function**
- 19. Tuples**

#

#

▼ Introduction

Python (programming language)

Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.[28]

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.[29]

Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python 2.0, released in 2000, introduced new features, such as list comprehensions, and a garbage collection system with reference counting. It was discontinued as version 2.7 in 2020.[30] Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, a free and open-source[31] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

History

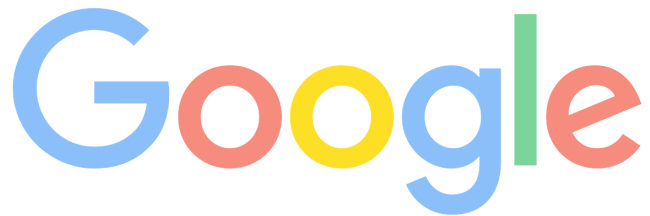
Python was conceived in the late 1980s[32] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL),[33] capable of exception handling and interfacing with the Amoeba operating system.[8] Its implementation began in December 1989.[34] Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator For Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker.[35] He now shares his leadership as a member of a five-person steering council.[36][37][38] In January 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.[39] Guido van Rossum has since then withdrawn his nomination for the 2020 Steering council.[40]

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.[41] [More](#)

▼ What You Can Do With Python



Instagram



Spotify®

NETFLIX



Dropbox



Pinterest



instacart



reddit

▼ My First Python Program

```
1 #Mi primer programa en Python
2 print("Hello World")
```

```
Hello World
```

▼ Variables

Variables are the names you give to computer memory locations which are used to store values in a computer program.

```
1 age = 20           #integer
2 price = 19.95      #Float
3 first_name = "Janner" #String
4 is_online = False  #Boolean
5
6 print(age)
7 print(price)
8 print(first_name)
9 print(is_online)
```

```
20
19.95
Janner
False
```

▼ Receiving Input

Input and output is terminology referring to the communication between a computer program and its user. Input is the user giving something to the program, and output is the program giving something to the user.

```
1 #input data (input)
```

```
2 name = input("What is your name? ")
3 #We join the variable "name" to a new string and print
4 print("Hello " + name)
```

```
What is your name? Janner
Hello Janner
```

▼ Type Conversion

In computer science, type conversion or typecasting refers to changing an entity of one datatype into another.

Ops! the program crash 😞

```
1 birth_year = input("Enter your birth year: ")
2 age = 2020 - birth_year #Here we have an Error!
3 # age = 2020 - "1981"
4 print(age)
```

```
Enter your birth year: 1981
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-20-368c2c04ef45> in <module>()
      1 birth_year = input("Enter your birth year: ")
----> 2 age = 2020 - birth_year #Here we have an Error!
      3 # age = 2020 - "1981"
      4 print(age)
```

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

SEARCH STACK OVERFLOW

Solution! 🧐

```
1 birth_year = input("Enter your birth year: ")
2 age = 2020 - int(birth_year)
3 print(age)
```

```
Enter your birth year: 1981
39
```

Functions For Conversions

```
int(): Integer data
```

```
float(): Decimal data
```

```
bool(): A binary variable, having two possible values called "true" and "false."
```

```
str(): Text strings
```

Practice

```
First: 10.1  
Second: 20  
Sum: 30.1
```

```
1 #Apparently something happens here!  
2 first = input("First: ")  
3 #first = "10"  
4 second = input("Second: ")  
5 #second = "20"  
6 sum = first + second  
7 #sum = "10" + "20"  
8 # "1020"  
9 print(sum)
```

```
First: 10  
Second: 20  
1020
```

```
1 #This is better!  
2 #Using int() function  
3 first = input("First: ")  
4 second = input("Second: ")  
5 sum = int(first) + int(second)  
6 print(sum)
```

```
First: 10  
Second: 20  
30
```

Ops! the program crash 🙄

```
1 first = input("First: ")
2 second = input("Second: ")
3 sum = int(first) + int(second)
4 print(sum)
```

First: 10.1
Second: 20

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-21-366f4dcbface> in <module>()
      1 first = input("First: ")
      2 second = input("Second: ")
----> 3 sum = int(first) + int(second)
      4 print(sum)
```

ValueError: invalid literal for int() with base 10: '10.1'

SEARCH STACK OVERFLOW

Solution! 🧐

```
1 #Using float() function
2 first = input("First: ")
3 second = input("Second: ")
4 sum = float(first) + float(second)
5 print("Sum: " + str(sum))
```

First: 10.1
Second: 20
Sum: 30.1

Other solution! 🧐

```
1 #Using float() function
2 first = input("First: ")
3 second = input("Second: ")
4 sum = float(first) + float(second)
5 print("Sum: " + str(sum))
```

▼ Strings

In computer programming, a string is traditionally a sequence of characters, either as a literal constant or as some kind of variable.

```
1 # Python for Beginners
2 # 0123456789..
3 course = 'Python for Beginners'
4
5 #text formatting
6 print(course.upper())
7 print(course.lower())
8 print(course)
9
10 #find
11 print(course.find('y'))
12 print(course.find('for'))
13
14 #replace
15 print(course.replace('for', '4'))
16
17 #boolean
18 print('Python' in course)
```

```
PYTHON FOR BEGINNERS
python for beginners
Python for Beginners
1
7
Python 4 Beginners
True
```

▼ Arithmetic Operators

The basic arithmetic operations are addition, subtraction, multiplication, and division. Arithmetic is performed according to an order of operations.

```
1 #sum
2 print(10 + 3)
3 #subtraction
4 print(10 - 3)
```

```
5 #division with decimals
6 print(10 / 3)
7 #division without decimals
8 print(10 // 3)
9 #module
10 print(10 % 3)
11 #exponentiation
12 print(10 ** 3)
13
14 #assignment operator
15 x = 10
16 x = x + 3
17 #x += 3
18 #x -= 3
19 #x *= 3
20 print(x)
```

```
13
7
3.3333333333333335
3
1
1000
13
```

▼ Operator Precedence

In mathematics and computer programming, the order of operations (or operator precedence) is a collection of rules that reflect conventions about which procedures to perform first in order to evaluate a given mathematical expression.

```
1 x = 10 + 3 * 2
2 x = (10 + 3) * 2
3 print(x)
```

```
26
```

▼ Comparison Operators

Operators that compare values and return true or false . The operators include: > , < , >= , <= , == , and != .

Haz doble clic (o pulsa Intro) para editar

```
1 x = 3 > 2          #greater than
2 #x = 3 >= 2        #greater than or equal to
3 #x = 3 < 2          #less than
4 #x = 3 <= 2         #less than or equal to
5 #x = 3 == 2         #equal to
6 #x = 3 != 2         #not equal to
7
8 print(x)
```

True

▼ Logical Operators

A logical operator is a symbol or word used to connect two or more expressions such that the value of the compound expression produced depends only on that of the original expressions and on the meaning of the operator.

and (both)

or (at least one)

not (negation)

```
1 price = 25
2 print(price > 10 and price < 30)
3 print(price > 10 or price < 30)
4 print(not price > 10)
```

True
True
False

▼ If Statements

The if statement allows you to control if a program enters a section of code or not based on whether a given condition is true or false. One of the important functions of the if statement is that it allows the program to select an action based upon the user's input.

```
1 #If condition
2 temperature = 35
3
4 if temperature > 30:
5     print("It's a hot day")
6     print("Drink plenty of water")
```

```
It's a hot day
Drink plenty of water
```

```
1 #;what happen here?
2 temperature = 25
3
4 if temperature > 30:
5     print("It's a hot day")
6     print("Drink plenty of water")
```

```
1 #Ok, is done!
2 temperature = 25
3
4 if temperature > 30:
5     print("It's a hot day")
6     print("Drink plenty of water")
7 print("Done")
```

```
Done
```

```
1 #elif / else
2 temperature = 25
3
4 if temperature > 30:
5     print("It's a hot day")
6     print("Drink plenty of water")
7 elif temperature > 20: # (21, 30]
8     print("It's a nice day")
9 elif temperature > 10: # (11, 20)
```

```
10 print("It's a bit cold")
11 else:
12     print("It's cold")
13 print("Done")
```

```
    It's a nice day
    Done
```

Exercise

```
Weight: 170
(K)g or (L)bs: 1
Weight in Kg: 76.5
```

Response

```
1 weight = int(input("Weight: "))
2 unit = input("(K)g or (L)bs: ")
3
4 if unit.upper() == "K":
5     converted = weight / 0.45
6     print("Weight in Lbs: " + str(converted))
7 else:
8     converted = weight * 0.45
9     print("Weight in Kgs: " + str(converted))
10
```

```
Weight: 170
(K)g or (L)bs: 1
Weight in Kgs: 76.5
```

▼ While Loops

In most computer programming languages, a while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.

```
1 print("1")
2 print("2")
3 print("3")
4 print("4")
```

```
1 print("5")  
5 print("5")
```

```
1 i = 1  
2 while i <= 5:  
3     print(i)  
4     i += 1
```

```
1  
2  
3  
4  
5
```

```
1 i = 1  
2 while i <= 10:  
3     print(i * '*')  
4     i += 1
```

```
*  
**  
***  
****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

▼ Lists

List is the most versatile data type available in functional programming languages used to store a collection of similar data items. The concept is similar to arrays in object-oriented programming. List items can be written in a square bracket separated by commas.

```
1  
1.1  
True  
'a'
```

```
1 names = ["John", "Bob", "Mosh", "Sam", "Mary"]
```

```
2 print(names)
3 print(names[0])
4 print(names[-1])
5 print(names[-2])
6 names[0] = "Jon"
7 print(names)
8 print(names[0:3])
```

```
['John', 'Bob', 'Mosh', 'Sam', 'Mary']
John
Mary
Sam
['Jon', 'Bob', 'Mosh', 'Sam', 'Mary']
['Jon', 'Bob', 'Mosh']
```

▼ List Methods

```
1 numbers = [1, 2, 3, 4, 5]
2 #The append() method appends an element to the end of the list.
3 numbers.append(6)
4 print(numbers)
5 #The list insert() method inserts an element to the list at the specified index.
6 numbers.insert(0, -1)
7 print(numbers)
8 #Remove() searches for the given element in the list and removes the first matching element
9 numbers.remove(3)
10 print(numbers)
11 #The clear() method removes all items from the list.
12 numbers.clear()
13 print(numbers)
```

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

```
1 numbers = [1, 2, 3, 4, 5]
2 print(1 in numbers)
3 #The len() function returns the number of items in an object
4 print(len(numbers))
```

```
True
5
```

▼ For Loops

In computer science, a for-loop (or simply for loop) is a control flow statement for specifying iteration, which allows code to be executed repeatedly. ... For-loops are typically used when the number of iterations is known before entering the loop.

```
1 numbers = [1, 2, 3, 4, 5]
2 print(numbers)
```

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

```
1 numbers = [1, 2, 3, 4, 5]
2 for item in numbers:
3     print(item)
```

```
1
2
3
4
5
```

```
1 i = 0
2 while i < len(numbers):
3     print(numbers[i])
4     i += 1
```

```
1
2
3
4
5
```

▼ The range() Function

Range() function generates a list of numbers between the given start integer to the stop integer.

```
1 numbers = range(5)
2 print(numbers)
```

```
range(0, 5)
```

```
1 for number in numbers:  
2     print(number)
```

```
0  
1  
2  
3  
4
```

```
1 numbers = range(5, 10)  
2 for number in numbers:  
3     print(number)
```

```
5  
6  
7  
8  
9
```

```
1 numbers = range(5, 10, 2)  
2 for number in numbers:  
3     print(number)
```

```
5  
7  
9
```

```
1 for number in range(5):  
2     print(number)
```

```
0  
1  
2  
3  
4
```

▼ Tuples

A tuple is a collection of objects which ordered and immutable

```
1 numbers = (1, 2, 3)  
2 numbers[0] = 10
```

```
TypeError                                Traceback (most recent call last)
<ipython-input-51-465a0d58e464> in <module>()
      1 numbers = (1, 2, 3)
----> 2 numbers[0] = 10
```

TypeError: 'tuple' object does not support item assignment

SEARCH STACK OVERFLOW

```
1 numbers = (1, 2, 3, 3)
2 print(numbers.count(3))
3 print(numbers.index(2))
```

```
2
1
```

Python Tutorial - Python for Beginners [2020]

[Reference link](#)