Python Tutorial

Prepared By
Eng./Ahmed Abd Elrahman

What is Python

Python is a versatile and beginner-friendly programming language that is widely used for web development, data analysis, artificial intelligence, and more. This tutorial will guide you through the basics of

It is used for:

- web development (server-side),software development, Data science
- Python can be used on a server to create web applications.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Python

Installing Python

Before you start coding, you need to have Python installed on your computer.- *Windows*:

Download the installer from the

[official Python website](https://www.python.org/downloads/), and follow the installation instructions.

Variable and Data Types

```
# Variables
x = 10
y = "Hello, World!“
z = 3.14


# Data Types
print(type(x))  # <class 'int'>
print(type(y))  # <class 'str'>
print(type(z))  # <class 'float'>
```

Comments

Comments in python

This is a single-line comment

"""This is

amulti-line

Comment

"""

Casting

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3


x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0


x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
```

Strings

- Strings in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is the same as "hello".

Strings are Arrays

```
a = "Hello, World!"
print(a[1])
```

- To get the length of a string, use the **len()** function.
- To check if a certain phrase or character is present in a string, we can use the keyword **in**.
- txt = "The best things in life are Water!"
  print(“Water" **in** txt) #True
-

- Booleans represent one of two values: **True** or **False**.

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

-

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Control Structures

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
else:
  print(“a is greater than b")
```

```
age = 18

if age >= 18:
print("You are an adult.")
elif age >= 13:
print("You are a teenager.")
else:
print("You are a child.")
```

Loops

#while

```
i = 1
while i < 6:
  print(i)
  i += 1
```

.break , With the break statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Loops

- **#continue,** With the continue statement we can stop the current iteration, and continue with the next:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Loops

- **#for loop**

```
for x in range(6):
  print(x)
```

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

```
for x in range(6):
  if x == 3: break
  print(x)
```

Functions

- In Python a function is defined using the **def** keyword:

```
def my_function():
  print("Hello from a function")
my_function()


def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Functions

```
def add(x):
  return 10 + x

print(add(3))
print(add(5))
print(add(9))
```

Lists

- Lists are used to store multiple items in a single variable.

```
List1= [“asd", “ali", “omar"]
print(List1)
print(List1[0]) # ’asd’
print(len(thislist)) #3
```

- List items can be of any data type:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

Lists

- A list can contain different data types:-

```
list1 = ["abc", 34, True, 40, "male"]
```

- To add an item to the end of the list, use the **append**() method

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

- The **remove**() method removes the specified item.

```
List2= ["apple", "banana", "cherry"]
List2.remove("banana")
print(List2)
```

Lists

- **Clear** the list content:

```
List2= ["apple", "banana", "cherry"]
List2.clear()
print(List2)
```

#Loop Through a List

```
List3= ["apple", "banana", "cherry"]
for x in  List3 :
  print(x)


List3 = ["apple", "banana", "cherry"]
for i in range(len(List3)):
  print(List3[i])
```

Dictionaries

- Dictionaries are used to store data values in key:value pairs.

```
Dict2= {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(Dict2[“year"])  # 1964
print(len(Dict2))
```

Dictionaries

 # Adding a new key-value pair

Dict2["email"] = ‘alice@example.com’

Looping through a dictionary

for key, value in Dict2.items():

  print(key+”,”+ value)

Array

- An array is a special variable, which can hold more than one value at a time.

```
cars = ["Ford", "Volvo", "BMW"]

print(len(cars))  # 3

print(cars[0])  # “Ford”

cars[0]=“Volly”

print(cars[0])  # “Volly”
```

 # Looping Array Elements

```
for x in cars:
  print(x)
```

Error Handling

- Use try-except blocks to handle errors gracefully.
- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

Error Handling

- **The try block will generate an exception, because x is not defined:**

```
try:
  print(x)
except:
  print("An exception occurred")
```

- The **finally** block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Modules and Packages

- Python has a rich ecosystem of libraries. You can import and use them in your code.

```
import math
print(math.sqrt(16))  # 4.0

# Importing specific functions
from math import pi
print(pi)  # 3.141592653589793
```

Input and Output

```
# Single variable
s = "Bob"
print(s)
# Multiple Variables
s = "Alice"
age = 25
city = "New York"
print(s, age, city)
```

Output

```
Bob
Alice 25 New York
```

Input and Output

```
# end Parameter with '@'
print("Python", end='@')
print("GeeksforGeeks")

# Seprating with Comma
print('G', 'F', 'G', sep='')

# for formatting a date
print('09', '12', '2016', sep='-')

# another example
print('pratik', 'geeksforgeeks', sep='@')
```

Output

```
Python@GeeksforGeeks
GFG
09-12-2016
pratik@geeksforgeeks
```

```
name = 'Tushar'
age = 23
print(f"Hello, My name is {name} and I'm {age} years old.")
```

Output

```
Hello, My name is Tushar and I'm 23 years old.
```

Input

Python **input()** function is used to take user input. By default, it returns the user input in form of a string.

Syntax: input(prompt)

- By default **input()** function helps in taking user input as **string**. If any user wants to take input as int or float, you just need to typecast it.

```
# Taking input as string
color = input("What color is rose?: ")
print(color)
```

Output

```
What color is rose?: Red
Red
```

Input

```
# Taking input as int
# Typecasting to int
n = int(input("How many roses?: "))
print(n)
```

Output

```
How many roses?: 8
8
```

```
# Taking input as float
# Typecasting to float
price = float(input("Price of each rose?: "))
print(price)
```

Output

```
Price of each rose?: 50.30
50.3
```

```
# Taking input from the user
num = int(input("Enter a value: "))

add = num + 5

# Output
print("The sum is %d" %add)
```

Output

```
Enter a value: 50The sum is 55
```

Take Multiple Input in Python

```
# taking two inputs at a time
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)

# taking three inputs at a time
x, y, z = input("Enter three values: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
print("Number of girls is : ", z)
```

Output

```
Enter two values: 5 10
Number of boys:  5
Number of girls:  10
Enter three values: 5 10 15
Total number of students:  5
Number of boys is :  10
Number of girls is :  15
```

Take Conditional Input from user in Python

```python
# Prompting the user for input
age_input = input("Enter your age: ")

# Converting the input to an integer
age = int(age_input)

# Checking conditions based on user input
if age < 0:
    print("Please enter a valid age.")
elif age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

Output

```
Enter your age: 22
You are an adult.
```

Object Oriented

- **How to create a class**

To define a class in Python, you can use the `class` keyword, followed by the class name and a colon. Inside the class, an `__init__` method has to be defined with `def`. This is the initializer that you can later use to instantiate objects. It's similar to a constructor in Java. `__init__` must always be present! It takes one argument: `self`, which refers to the object itself. Inside the method, the `pass` keyword is used as of now, because Python expects you to type something there. Remember to use correct indentation!

```
class Dog:

    def __init__(self):
        pass
```

- **self** parameter is a reference to the current instance of the class. It allows us to access the attributes and methods of the object.
- To instantiate an object, type the class name, followed by two brackets. You can assign this to a variable to keep track of the object.

D1 = Dog()

Adding attributes to a class

```python
class Dog:
    species = "Canine"  # Class attribute

    def __init__(self, name, age):
        self.name = name  # Instance attribute
        self.age = age  # Instance attribute
```

Explanation:

- **`class Dog`**: Defines a class named `Dog`.
- **`species`**: A class attribute shared by all instances of the class.
- **`__init__` method**: Initializes the `name` and `age` attributes when a new object is created.

Example

```
class Dog:

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
ozzy = Dog(“Bravo", 2)

print(ozzy.name)
print(ozzy.age)
```

```
Bravo
2
```

Attributes Types

- **Public Attributes**: Can be accessed and modified from anywhere.
- **Protected Attributes**: Indicated by a **single underscore** (_attribute). This is a convention that suggests the attribute should not be accessed directly but is still accessible.
- **Private Attributes**: Indicated by a **double underscore** (__attribute). Python performs name mangling, making it harder to access from outside the class.

Example Attributes Types

```python
class Person:
    def __init__(self, public_attr, protected_attr, private_attr):
        self.public_attr = public_attr          # Public attribute
        self._protected_attr = protected_attr   # Protected attribute
        self.__private_attr = private_attr      # Private attribute

    # Getter for private attribute
    def get_private_attr(self):
        return self.__private_attr

    # Setter for private attribute
    def set_private_attr(self, value):
        self.__private_attr = value

    # Getter for protected attribute
    def get_protected_attr(self):
        return self._protected_attr

    # Setter for protected attribute
    def set_protected_attr(self, value):
        self._protected_attr = value

# Testing the class
person = Person("Public Value", "Protected Value", "Private Value")

# Accessing public attribute
print("Public Attribute:", person.public_attr)  # ✅ Allowed

# Accessing protected attribute (not recommended)
print("Protected Attribute:", person.get_protected_attr())  # ✅ Allowed but discouraged

# Accessing private attribute (only via getter method)
print("Private Attribute:", person.get_private_attr())  # ✅ Allowed via getter
```

Example 2

```
class Dog:
    species = "Canine"  # Class attribute

    def __init__(self, name, age):
        self.name = name  # Instance attribute
        self.age = age  # Instance attribute

# Creating an object of the Dog class
dog1 = Dog("Buddy", 3)

print(dog1.name)
print(dog1.species)
```

Output

```
Buddy
Canine
```

Define methods in a class

```
class Dog:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        print("bark bark!")

    def doginfo(self):
        print(self.name + " is " + str(self.age) + " year(s) old.")

ozzy = Dog("Ozzy", 2)
skippy = Dog("Skippy", 12)
filou = Dog("Filou", 8)

ozzy.doginfo()
skippy.doginfo()
filou.doginfo()
```

Ozzy is 2 year(s) old.
Skippy is 12 year(s) old.
Filou is 8 year(s) old.

Full Example

```
class Person:
    def __init__(self, nage, age):
        self._nage = nage  # Using a single underscore to indicate a 'protected' attribute
        self._age = age

    # Getter for nage
    def get_nage(self):
        return self._nage

    # Setter for nage
    def set_nage(self, new_nage):
        self._nage = new_nage

    # Getter for age
    def get_age(self):
        return self._age

    # Setter for age
    def set_age(self, new_age):
        self._age = new_age

# Testing the class
person1 = Person("John", 25)
person2 = Person("Alice", 30)

# Using getters
print("Person 1 Name:", person1.get_nage())
print("Person 1 Age:", person1.get_age())

print("Person 2 Name:", person2.get_nage())
print("Person 2 Age:", person2.get_age())

# Using setters
person1.set_nage("Johnny")
person1.set_age(26)
print("Updated Person 1 Name:", person1.get_nage())
print("Updated Person 1 Age:", person1.get_age())
```