**CHALLENGE DAY - 1**

**(1/100)**

**AUTHOR : ANAND VEERARAHAVAN**

**OBJECTIVE :**

**To research and complete all the tasks given below**

**GOALS :**

**1. PYTHON SYNTAX**

**2. KEYWORDS**

**3. VARIABLES**

**4. DATATYPES**

**5. INPUT AND OUTPUT**

**6. OPERATORS**

**7. CONTROL FLOW**

**8. LOOPS**

**9. LIST**

**10. TUPLE**

**11. DICTIONARY**

**12. SETS**

**13. STRINGS**

GOAL - 1

**PYTHON SYNTAX**

First Python Program

Let us execute the programs in different modes of programming.

Interactive Mode Programming

Step 1: Open Anaconda Powershell Prompt

Step 2: Type python

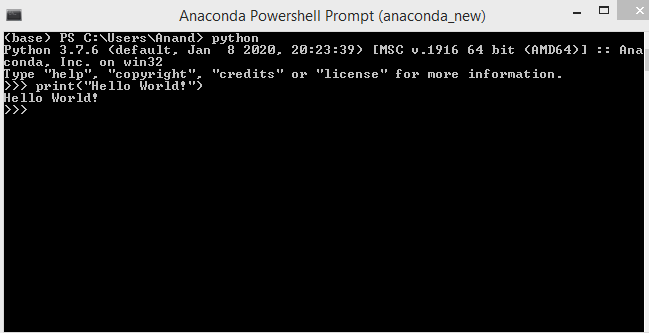
Here, you will get the version of python installed in your pc.

Step 3: When the cursor shows ">>" , type a print statement

For example : >> print("Hello world!")

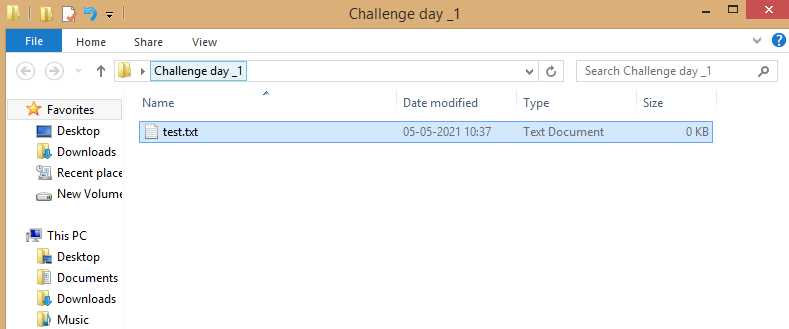
Refer to the screenshot for more help.

Invoking the interpreter without passing a script file as a parameter brings up the following prompt −



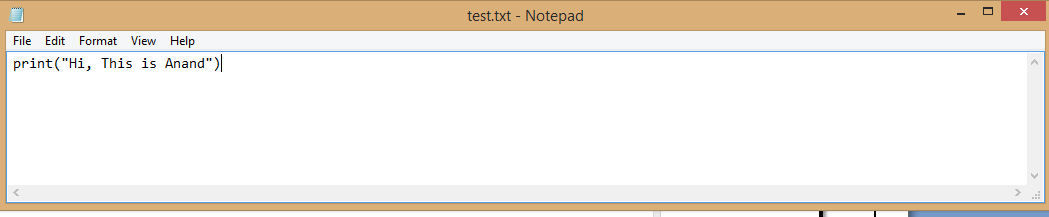
### Script Mode Programming

STEP 1: Create a folder and inside the folder create a text file called "test.txt"



STEP 2: Open the file and type the print statement

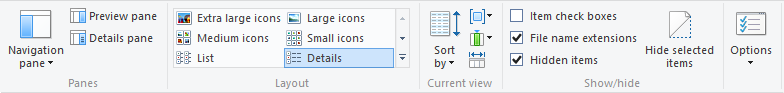
print("Hi, This is Anand.")

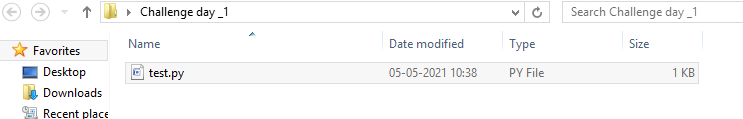


Save the file and close it.

STEP 3: Go to the view in the folder and select the checkbox of "File name extensions".

Now click on your file and rename the file to "test.py"





STEP 5: Copy your folder path

For example : C:\Users\Anand\Desktop\Challenge day \_1

STEP 6 : Open Anaconda Prompt and type cd(change directory) and then the folder path.

For example :

cd C:\Users\Anand\Desktop\Challenge day \_1

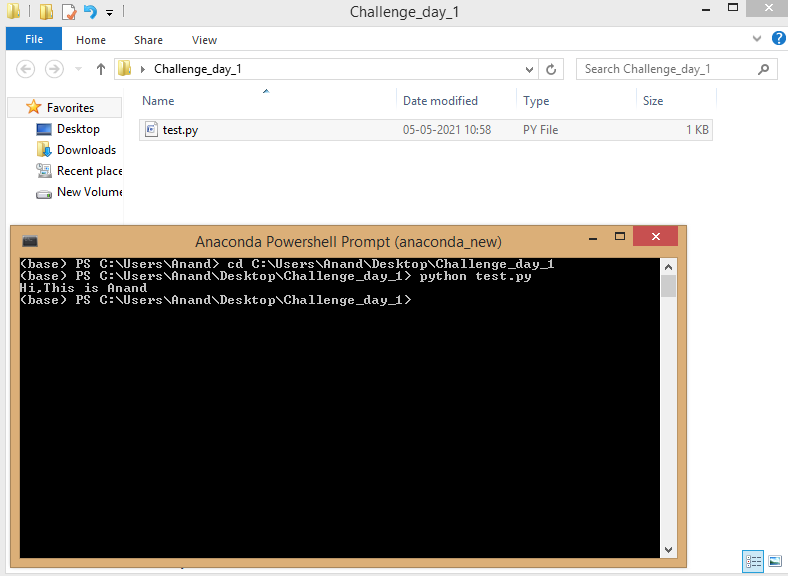
STEP 7: After the path has changed type :

python test.py to view the output.

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have the extension **.py**. Type the following source code in a test.py file −

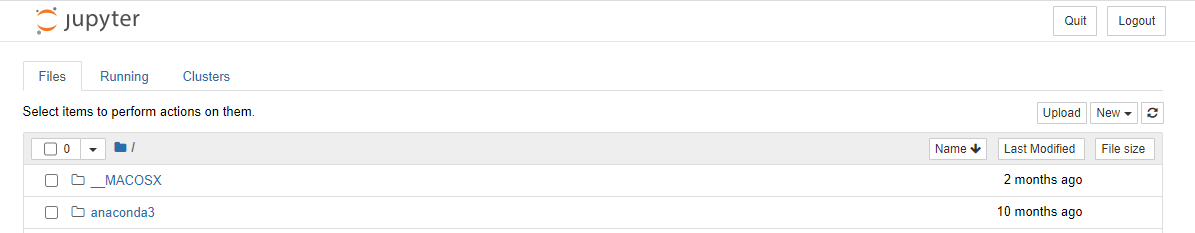
See the results below:



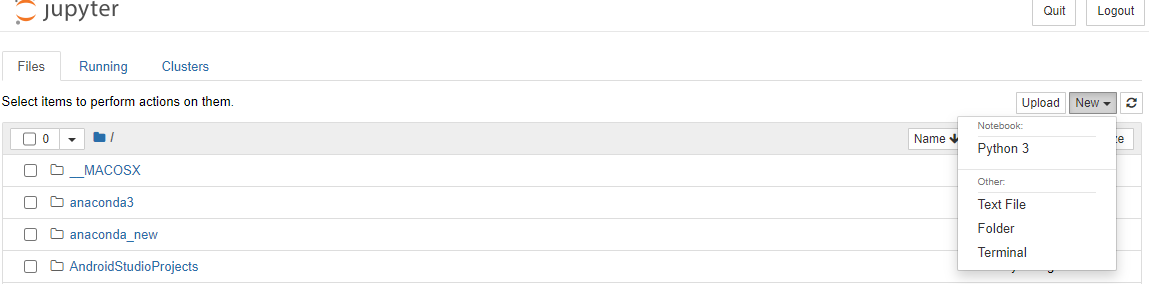
**IDE Programming:**

STEP 1 : Open Jupyter notebook on your pc.

STEP 2 : A home page is shown. On the right hand side top of the panel, you can see "New" Button as shown below.

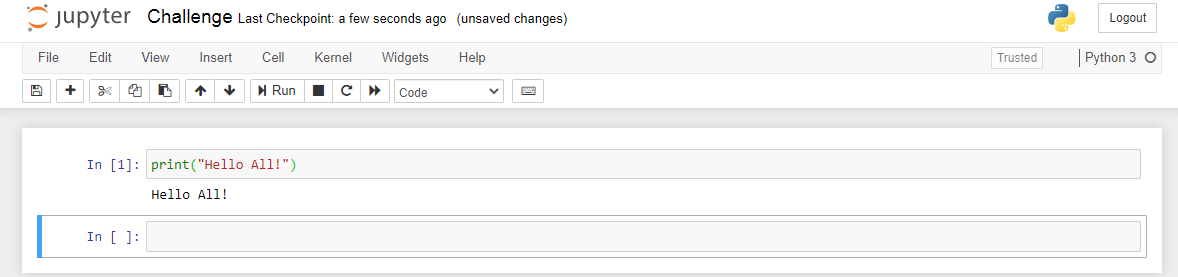


STEP 3: Click on "Python 3" to create a python file.



STEP 4: Type the code print("Hello All!") in the cell and click on run to view the output

For example :



**GOAL - 2**

**KEYWORDS**

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

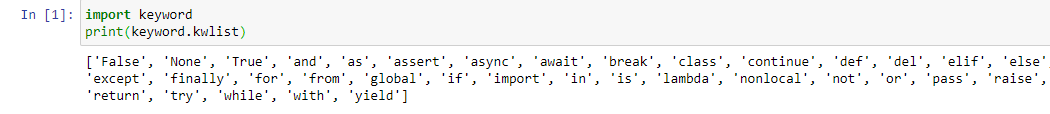
STEP 1: To find the key word list , open a python file in Jupyter notebook.

STEP 2: Type :

import keyword

print(keyword.kwlist)

For example :



Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifier

**GOAL - 3**

**VARIABLES**

Variables

Variables are containers for storing data value

Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example

x = 5  
y = "John"  
print(x)  
print(y)

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

Example

x = 4       # x is of type int  
x = "Sally" # x is now of type str  
print(x)

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

x = str(3)    # x will be '3'  
y = int(3)    # y will be 3  
z = float(3)  # z will be 3.0

Get the Type

You can get the data type of a variable with the type() function.

Example

x = 5  
y = "John"  
print(type(x))  
print(type(y))

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

Example

x = "John"  
# is the same as  
x = 'John'

Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

a = 4  
A = "Sally"  
#A will not overwrite a

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume). Rules for Python variables:

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

Legal variable names:

myvar = "John"  
my\_var = "John"  
\_my\_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"

Example

Illegal variable names:

2myvar = "John"  
my-var = "John"  
my var = "John"

Remember that variable names are case-sensitive

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

Camel Case

Each word, except the first, starts with a capital letter:

myVariableName = "John"

Pascal Case

Each word starts with a capital letter:

MyVariableName = "John"

Snake Case

Each word is separated by an underscore character:

my\_variable\_name = "John"

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)

**Note:** Make sure the number of variables matches the number of values, or else you will get an error.

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

x = y = z = "Orange"  
print(x)  
print(y)  
print(z)

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

fruits = ["apple", "banana", "cherry"]  
x, y, z = fruits  
print(x)  
print(y)  
print(z)

Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

Example

x = "awesome"  
print("Python is " + x)

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_variables3)

You can also use the + character to add a variable to another variable:

Example

x = "Python is "  
y = "awesome"  
z =  x + y  
print(z)

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_variables4)

For numbers, the + character works as a mathematical operator:

Example

x = 5  
y = 10  
print(x + y)

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_variables5)

If you try to combine a string and a number, Python will give you an error:

Example

x = 5  
y = "John"  
print(x + y)

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

Create a variable outside of a function, and use it inside the function

x = "awesome"  
  
def myfunc():  
  print("Python is " + x)  
  
myfunc()

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

x = "awesome"  
  
def myfunc():  
  x = "fantastic"  
  print("Python is " + x)  
  
myfunc()  
  
print("Python is " + x)

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

x = "awesome"  
  
def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

**GOAL - 4**

**DATATYPES**

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

|  |  |
| --- | --- |
| Text Type: | str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | dict |
| Set Types: | set, frozenset |
| Boolean Type: | bool |
| Binary Types: | bytes, bytearray, memoryview |

Getting the Data Type

You can get the data type of any object by using the type() function:

Example

Print the data type of the variable x:

x = 5  
print(type(x))

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** | **Try it** |
| x = "Hello World" | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str) |
| x = 20 | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int) |
| x = 20.5 | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float) |
| x = 1j | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex) |
| x = ["apple", "banana", "cherry"] | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list) |
| x = ("apple", "banana", "cherry") | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple) |
| **Example** | **Data Type** | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range) |
| x = {"name" : "John", "age" : 36} | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict) |
| x = {"apple", "banana", "cherry"} | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set) |
| x = frozenset({"apple", "banana", "cherry"}) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset) |
| x = True | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool) |
| x = b"Hello" | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview) |

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** | **Try it** |
| x = str("Hello World") | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str2) |
| x = int(20) | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int2) |
| x = float(20.5) | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float2) |
| x = complex(1j) | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex2) |
| x = list(("apple", "banana", "cherry")) | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list2) |
| x = tuple(("apple", "banana", "cherry")) | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple2) |
| x = range(6) | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range2) |
| x = dict(name="John", age=36) | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict2) |
| x = set(("apple", "banana", "cherry")) | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set2) |
| x = frozenset(("apple", "banana", "cherry")) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset2) |
| x = bool(5) | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool2) |
| x = bytes(5) | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes2) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray2) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview2) |

**GOAL - 5**

**OPERATORS**

Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

Example

print(10 + 5)

[Run example »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper)

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| + | Addition | x + y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_add) |
| - | Subtraction | x - y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_sub) |
| \* | Multiplication | x \* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mult) |
| / | Division | x / y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_div) |
| % | Modulus | x % y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mod) |
| \*\* | Exponentiation | x \*\* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_exp) |
| // | Floor division | x // y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_floordiv) |

Python Assignment Operators

Assignment operators are used to assign values to variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass6) |
| //= | x //= 3 | x = x // 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass7) |
| \*\*= | x \*\*= 3 | x = x \*\* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass8) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass9) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass10) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass11) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass12) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass13) |

Python Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal | x == y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare4) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare5) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare6) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare7) |

Python Logical Operators

Logical operators are used to combine conditional statements:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| and | Returns True if both statements are true | x < 5 and  x < 10 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical1) |
| or | Returns True if one of the statements is true | x < 5 or x < 4 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical2) |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical3) |

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| is | Returns True if both variables are the same object | x is y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity1) |
| is not | Returns True if both variables are not the same object | x is not y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity2) |

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| in | Returns True if a sequence with the specified value is present in the object | x in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership1) |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership2) |

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left,  and let the rightmost bits fall off |

**GOAL - 6**

**CONTROL FLOW**

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

* 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

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:

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

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

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

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_error)

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_elif)

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

Else

The else keyword catches anything which isn't caught by the preceding conditions.

Example

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")  
else:  
  print("a is greater than b")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_else)

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

Example

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

if a > b: print("a is greater than b")

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

a = 2  
b = 330  
print("A") if a > b else print("B")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_if_else_short)

This technique is known as **Ternary Operators**, or **Conditional Expressions**.

You can also have multiple else statements on the same line:

Example

One line if else statement, with 3 conditions:

a = 330  
b = 330  
print("A") if a > b else print("=") if a == b else print("B")

And

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

a = 200  
b = 33  
c = 500  
if a > b and c > a:  
  print("Both conditions are True")

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

a = 200  
b = 33  
c = 500  
if a > b or a > c:  
  print("At least one of the conditions is True")

Nested If

You can have if statements inside if statements, this is called *nested* if statements.

Example

x = 41  
  
if x > 10:  
  print("Above ten,")  
  if x > 20:  
    print("and also above 20!")  
  else:  
    print("but not above 20.")

The pass Statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

Example

a = 33  
b = 200  
  
if b > a:  
  pass

Python Loops

Python has two primitive loop commands:

* while loops
* for loops

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

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

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_while)

**Note:** remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement

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

Example

Exit the loop when i is 3:

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

The continue Statement

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

Example

Continue to the next iteration if i is 3:

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

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

i = 1  
while i < 6:  
  print(i)  
  i += 1  
else:  
  print("i is no longer less than 6")

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

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

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_for)

The for loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

for x in "banana":  
  print(x)

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when x is "banana":

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)  
  if x == "banana":  
    break

Example

Exit the loop when x is "banana", but this time the break comes before the print:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    break  
  print(x)

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example

Do not print banana:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    continue  
  print(x)

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the range() function:

for x in range(6):  
  print(x)

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_for_range)

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Example

Using the start parameter:

for x in range(2, 6):  
  print(x)

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_for_range2)

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

Example

Increment the sequence with 3 (default is 1):

for x in range(2, 30, 3):  
  print(x)

Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

for x in range(6):  
  print(x)  
else:  
  print("Finally finished!")

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_for_else)

**Note:** The else block will NOT be executed if the loop is stopped by a break statement.

Example

Break the loop when x is 3, and see what happens with the else block:

for x in range(6):  
  if x == 3: break  
  print(x)  
else:  
  print("Finally finished!")

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

Print each adjective for every fruit:

adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]  
  
for x in adj:  
  for y in fruits:  
    print(x, y)

The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

Example

for x in [0, 1, 2]:  
  pass

**GOAL - 7**

**LIST**

# Python List

mylist = ["apple", "banana", "cherry"]

## List

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

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

### Example

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

## Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

**Note:** There are some [list methods](https://www.w3schools.com/python/python_lists_methods.asp) that will change the order, but in general: the order of the items will not change.

## Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

## Allow Duplicates

Since lists are indexed, lists can have items with the same value:

### Example

Lists allow duplicate values:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)

## List Length

To determine how many items a list has, use the len() function:

### Example

Print the number of items in the list:

thislist = ["apple", "banana", "cherry"]  
print(len(thislist))

## List Items - Data Types

List items can be of any data type:

### Example

String, int and boolean data types:

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

A list can contain different data types:

### Example

A list with strings, integers and boolean values:

list1 = ["abc", 34, True, 40, "male"]

## type()

From Python's perspective, lists are defined as objects with the data type 'list':

<class 'list'>

### Example

What is the data type of a list?

mylist = ["apple", "banana", "cherry"]  
print(type(mylist))

## The list() Constructor

It is also possible to use the list() constructor when creating a new list.

### Example

Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered and unindexed. No duplicate members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\* and changeable. No duplicate members.

For List Operations refer this website :

https://www.w3schools.com/python/python\_lists.asp

**GOAL - 8**

**TUPLE**

# Python Tuples

mytuple = ("apple", "banana", "cherry")

## Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

### Example

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

## Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

## Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

## Allow Duplicates

Since tuple are indexed, tuples can have items with the same value:

### Example

Tuples allow duplicate values:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)

## Tuple Length

To determine how many items a tuple has, use the len() function:

### Example

Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))

## Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

### Example

One item tuple, remember the commma:

thistuple = ("apple",)  
print(type(thistuple))  
  
#NOT a tuple  
thistuple = ("apple")  
print(type(thistuple))

## Tuple Items - Data Types

Tuple items can be of any data type:

### Example

String, int and boolean data types:

tuple1 = ("apple", "banana", "cherry")  
tuple2 = (1, 5, 7, 9, 3)  
tuple3 = (True, False, False)

A tuple can contain different data types:

### Example

A tuple with strings, integers and boolean values:

tuple1 = ("abc", 34, True, 40, "male")

## type()

From Python's perspective, tuples are defined as objects with the data type 'tuple':

<class 'tuple'>

### Example

What is the data type of a tuple?

mytuple = ("apple", "banana", "cherry")  
print(type(mytuple))

## The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

### Example

Using the tuple() method to make a tuple:

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets  
print(thistuple)

**GOAL - 9**

**DICTIONARIES**

# Python Dictionaries

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}

## Dictionary

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

A dictionary is a collection which is ordered\*, changeable and does not allow duplicates.

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

Dictionaries are written with curly brackets, and have keys and values:

### Example

Create and print a dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)

## Dictionary Items

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

### Example

Print the "brand" value of the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict["brand"])

## Ordered or Unordered?

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

## Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

## Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

### Example

Duplicate values will overwrite existing values:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964,  
  "year": 2020  
}  
print(thisdict)

## Dictionary Length

To determine how many items a dictionary has, use the len() function:

### Example

Print the number of items in the dictionary:

print(len(thisdict))

## Dictionary Items - Data Types

The values in dictionary items can be of any data type:

### Example

String, int, boolean, and list data types:

thisdict = {  
  "brand": "Ford",  
  "electric": False,  
  "year": 1964,  
  "colors": ["red", "white", "blue"]  
}

## type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

<class 'dict'>

### Example

Print the data type of a dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(type(thisdict))

**GOAL - 10**

**SETS**

myset = {"apple", "banana", "cherry"}

Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Tuple](https://www.w3schools.com/python/python_tuples.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A set is a collection which is both *unordered* and *unindexed*.

Sets are written with curly brackets.

Example

Create a Set:

thisset = {"apple", "banana", "cherry"}  
print(thisset)

**Note:** Sets are unordered, so you cannot be sure in which order the items will appear.

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Sets are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can add new items.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Example

Duplicate values will be ignored:

thisset = {"apple", "banana", "cherry", "apple"}  
  
print(thisset)

Get the Length of a Set

To determine how many items a set has, use the len() method.

Example

Get the number of items in a set:

thisset = {"apple", "banana", "cherry"}  
  
print(len(thisset))

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

set1 = {"apple", "banana", "cherry"}  
set2 = {1, 5, 7, 9, 3}  
set3 = {True, False, False}

A set can contain different data types:

Example

A set with strings, integers and boolean values:

set1 = {"abc", 34, True, 40, "male"}

type()

From Python's perspective, sets are defined as objects with the data type 'set':

<class 'set'>

Example

What is the data type of a set?

myset = {"apple", "banana", "cherry"}  
print(type(myset))

The set() Constructor

It is also possible to use the set() constructor to make a set.

Example

Using the set() constructor to make a set:

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets  
print(thisset)

**GOAL - 11**

**STRINGS**

Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

print("Hello")  
print('Hello')

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

a = "Hello"  
print(a)

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

Or three single quotes:

Example

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

**Note:** in the result, the line breaks are inserted at the same position as in the code.

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

a = "Hello, World!"  
print(a[1])

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Example

Loop through the letters in the word "banana":

for x in "banana":  
  print(x)

String Length

To get the length of a string, use the len() function.

Example

The len() function returns the length of a string:

a = "Hello, World!"  
print(len(a))

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example

Check if "free" is present in the following text:

txt = "The best things in life are free!"  
print("free" in txt)

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_is_in)

Use it in an if statement:

Example

Print only if "free" is present:

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

Example

Check if "expensive" is NOT present in the following text:

txt = "The best things in life are free!"  
print("expensive" not in txt)

Use it in an if statement:

Example

print only if "expensive" is NOT present:

txt = "The best things in life are free!"  
if "expensive" not in txt:  
  print("Yes, 'expensive' is NOT present.")

**RESULT :**

**Thus all the goals of the day have been completed with extensive research.**

**REFERENCES :**

**https://www.w3schools.com/python/**

**https://www.tutorialspoint.com/python/python\_basic\_syntax.htm**

**https://www.tutorialsteacher.com/python/python-syntax**

**https://stackoverflow.com/**