

# Pythonic

Mastering the Art



of Python Programming  
for beginners

Programming is the process of writing computer programs using a specific programming language. Programming allows creating applications and tools that help solve problems and optimize processes in many different areas such as business, medicine, science, entertainment, among others.

One of the most popular programming languages currently is Python. Python was created in 1991 by Guido van Rossum and is a popular and easy-to-use programming language. They are used in many different applications and fields such as education, data analysis, and web application development.

Python has many great features that make it popular among programmers. Some of these features include:

Ease of use: Python has a simple, clean structure and an easy-to-understand programming language, which makes it ideal for beginners and professionals alike.

Open source: Python is open source and this means that developers can access, modify, improve and expand the code of the basic language to suit their own needs.

Reliability and security: Python provides security and reliability features, offers tools to prevent common programming errors.

Productivity: Python is characterized by high productivity because it allows programmers to write quickly and with a minimum amount of code.

Python is used in many different fields and applications such as web application development, data analysis, and manifestology

# Print

The print function in Python is used to print text or a certain value to the screen.

They can be used to show text or a value to the user, or to check the value of a particular variable during program execution.

To use the print function, the "print" command is used, and the text or value that you want to print is passed as an intermediate to the function. For example, to print the text "Hello, World!" In Python, the following command can be used

```
print("Hello, World!")
```

The print function can also be used to print the value of a specific variable, as in the following example:

```
x = 10  
print(x)
```

The result «10» will appear on the screen.

The function can also be used to print several values at the same time, such as:

```
print("Hello,", "World!")
```

The result will show «Hello, World!» On the screen.

# Varibales

In the Python language, variables are used to store values and data that are used in the program. It can be said that variables represent containers for data, and are used to permanently preserve values during the execution of the program.

The definition of variables in Python does not require any specific type for the data stored in the variable, that is, the program does not need to specify the type of the variable when defining it, and the type of the variable can be changed at any time during the execution of the program. For example, the variable « x » with a value of «5» can be defined as follows:

```
x = 5
```

After that, the value of the variable «x» can be changed to «10», as follows:

```
x = 10
```

Multiple variables can also be defined in Python, as in the following example:

```
x = 5  
y = "Hello, World!"  
z = 3.14
```

Any value can be assigned to any variable, be it a numerical value, text, or of any other type. Variables can also be used in other calculations or code, as in the following example:

```
x = 5  
y = 10  
z = x + y  
print(z)
```

The result «15» will appear on the screen. In this example, the variables «x» and «y» were used in the calculation «x + y», and the result was stored in the variable «z». And then the print function was used to print the value of the variable «z» on the screen.

# String

In Python, a «string» is defined as a set of consecutive characters that are represented between regular quotation marks («) or double quotation marks («»). Strings can be used to represent texts and other textual information in the program.

For example, a simple string can be defined in the following way:

```
my_string = 'Hello, world!'
```

Or by double quoting as follows:

```
my_string = "Hello, world!"
```



An empty string can also be defined simply as follows:

```
empty_string = ''
```

Or define a string containing specific characters using indexing as follows:

```
my_string = 'Hello, world!'
first_char = my_string[0]
```

# Numbers

int,float,complex

In Python, int is used to represent integers. A variable of this type can be created using the following general form:

```
variable_name = integer_value
```

Where:

variable\_name is the name of the variable you want to create.

integer\_value is the correct value that you want to store in the variable.

For example, the number 5 can be stored in a variable of type int using the following code:

```
my_int = 5
```

The value of the variable can be printed using the print () function as follows:

```
print(my_int)
```

The result will appear on the screen as follows:

```
5
```

Note: basic arithmetic operations (such as addition, subtraction, multiplication and division) can be used on variables of type int and can also be converted to other types in case of need.

In Python, float is a type of numeric data used to represent small and large real numbers. The term «float» implies that these numbers float on the surface (floating), that is, they move between integers and those that have decimal digits.

Floating numbers are distinguished by their ability to represent large, small and decimal numbers with high accuracy, and are also widely used in scientific, engineering and financial calculations.

```
x = 3.14
```

In this example, the variable x is defined with a floating decimal value equal to 3.14

Complex numbers are numbers that consist of a real part and an imaginary part and are defined in Python using the symbol  $j$  to represent the imaginary part.

A complex number in Python can be created using the `complex()` function, which takes one or two parameters to determine the value of the real part and the imaginary part. For example, the complex number  $2 + 3j$  can be constructed as follows:

```
x = complex(3, 2)
print(x)  # (3+2j)
```

Complex numbers can also be created using mixed numbers, which have a real part and an imaginary part. For example, the complex number  $4 + 1j$  can be defined as follows:

```
x = 1 + 4j
print(x)  # (1+4j)
```

Python supports many arithmetic operations on complex numbers, such as addition, subtraction, multiplication and division, and these operations can be done using the usual arithmetic coefficients, such as  $+$ ,  $-$ ,  $*$  and  $/$ .

# Comments

Comments in Python are used to write text that explains or clarifies part of the code, and is not executed when the program is run. Comments can be used for several reasons, such as:

- Explaining the code to other readers.

- Documenting the code.

- Ignoring parts of the code for testing new things.

A comment in Python can be created using the hash symbol (#) when it is placed at the beginning of a line.

After the hash symbol, the text to be explained is written, as follows:

```
# This is a comment in Python
```

Comments can also be placed on the same line as the code, by placing the hash symbol at the beginning of the text to be explained, as follows:

```
x = 5 # This is a comment that explains the value of x
```

Comments can be written in any programming language, but they should be in English or the official language in the country where Python is used for official or public projects.

# Comparison operations

In Python, comparison operations are performed using special comparison operators, namely:

Equality operator ( `=` ) - compares the two values on the sides of the operator and returns True if the two values are equal, and False if the two values are unequal.

The inequality operator ( `!=` )- It compares the two values on the sides of the operator and returns True if the two values are unequal, and False if the two values are equal.

Operator greater than ( `>` ) - compares the two values on the sides of the operator and returns True if the first value is greater than the second value, and False if the first value is smaller than the second value.

Operator smaller than ( `<` ) - compares the two values on the sides of the operator and returns True if the first value is smaller than the second value, and False if the first value is larger than the second value.

Operator greater than or equal to ( `>=` ) - compares the two values on both sides of the operator and returns True if the first value is greater than or equal to the second value, and False if the first value is smaller than the second value.

Operator smaller than or equal to ( `<=` ) - compares the two values on the sides of the operator and returns True if the first value is smaller than or equal to the second value, and False if the first value is greater than the second value.

Operations are carried out

# Booleans

In Python, bool is a built-in type of data that represents true and false values. It is used to represent logical decisions and logical conditions in programming.

It takes a True value if the analyzed value is true, and takes a False value if the value is incorrect.

For example, the variable x can be defined as having a True value as follows:

```
x = True
```

Or the Y variable can be defined as having a False value as follows:

```
y = False
```

The True and False values can be used to create conditions in programming. For example, if we have a variable z that has the value 5, we can write a condition that checks whether z is equal to 5 as follows:

```
if z == 5:  
    print("z equals 5")
```

The print command will be executed only if the condition is true, this means that the True value has been used in the condition.



# Mathematical operations

Arithmetic operations in Python can be done using the basic arithmetic operations supported by the language. Examples of such operations include:

Plural (+)

Subtract (-)

Hit (\*)

Divide (/)

Remainder of the division (%)

Ranking strength ( \*\* )

For example, you can perform a combination of two numbers in Python as follows:

```
a = 5
b = 3
c = a + b
print(c)
```

You can use other calculations in the same way. And in case you want to do a more complex calculation, arithmetic libraries can be used in Python, such as NumPy and pandas.

You can use the subtraction operation in Python using the symbol « - » as follows:

```
a = 10  
b = 5  
c = a - b  
print(c)
```

In this example, two variables a and b were created and combined using the subtraction operation, then the result was stored in variable c, and finally the value of c was printed using the print () function.

You can use this operation with any type of Number, be it integers or decimals. They can also be used with variables that contain numeric values.

You can use the multiplication operation in Python using the symbol « \* » as follows:

```
a = 3
b = 4
c = a * b
print(c)
```

In this example, two variables a and b were created and multiplied using the multiplication operation, then the result was stored in variable c, and finally the value of c was printed using the print () function.

You can use this operation with any type of Number, be it integers or decimals. They can also be used with variables that contain numeric values.

You can use the division operation in Python using the symbol «/» as follows:

```
a = 10  
b = 2  
c = a / b  
print(c)
```

In this example, two variables a and b were created and divided a by b using the division operation, then the result was stored in variable c, and finally the value of c was printed using the print () function.

When using the division operation, a result is returned in decimal form, even if the number by which it is divided and the product are integers. If you want to get a correct result, you can use the operation «//» instead of «/», and this operation will return only a correct result. For example:

```
a = 10
b = 3
c = a // b
print(c) # 3
```

In this example, the operation «//» was used instead of «/», which returned an integer value.

In Python, you can get the remainder of the division using the mathematical sign%, and it returns the remainder of the division between the two given numbers.

For example, if you want to get the remainder of the division at a division of  $3 \div 7$ , you can use the percentage sign as follows:

```
>>> 7 % 3
1
```

And here the result is 1 because the remainder of the division is 1, where it can be written as:

$$7 = (3 \times 2) + 1$$

The remainder of the division can be used in many applications, such as determining whether a certain number is odd or even, determining which week corresponds to the specified date, etc.

# Input

`input ()` is a function built into Python that allows the user to enter data through the keyboard. This function takes a text (it can be used as an educational text) and prints it on the screen, after which it waits for the data to be entered by the user.

When the user enters data through `input ()`, the value entered by the user is saved as text. This value can be stored in a variable for use in the rest of the program.

`Input()` can be used in many applications, such as creating a program that asks the user to enter their name, or a program that asks the user to enter a number. In both cases, `input()` is used to get the input that the user enters.

In Python, the input value (input) is always considered text (string). This means that the value that the user enters will be saved as text.

The input value can be read from the user using input() and then stored in a variable, as follows:

```
name = input("What is your name? ")
```

In this example, the text «What is your name?» On the screen, the user is asked to enter his name. The name entered by the user is saved in the name variable. And this value is stored as text.

The variable name in the program can then be used as text, as in the following examples:

```
print("Hello, " + name + "!")
```

Using the + operation, the text «Hello,» can be combined with the value of the variable name and then with the text»!» To create a welcome sentence that uses the username he entered.



To take an integer input in Python, you can use the `input()` function to get the input as a string and then convert it to an integer using the `int()` function. Here's an example:

```
num = int(input("Enter an integer: "))
```

In this example, the `input()` function prompts the user to enter an integer, which is then stored as a string in the `num` variable. The `int()` function is used to convert the string to an integer.

Note that if the user enters a non-integer value, this will result in a `ValueError`. You can use a `try-except` block to handle this error if needed.

To take a floating-point number input in Python, you can use the `input()` function to get the input as a string and then convert it to a float using the `float()` function. Here's an example:

```
num = float(input("Enter a floating-point number: "))
```

In this example, the `input()` function prompts the user to enter a floating-point number, which is then stored as a string in the `num` variable. The `float()` function is used to convert the string to a floating-point number.

Note that if the user enters a non-numeric value, this will result in a `ValueError`. You can use a `try-except` block to handle this error if needed.

# If Condition

if is a conditional statement in Python used to execute a block of code only if a certain condition is true. The syntax for the if statement is as follows:

```
if condition:  
    # execute code if condition is true
```

Here, condition is a Boolean expression that is evaluated to either True or False. If condition is True, then the code inside the block is executed, and if condition is False, then the code is skipped.

Optionally, an else clause can be added to the if statement to execute a different block of code if the condition is False:

```
if condition:  
    # execute code if condition is true  
else:  
    # execute code if condition is false
```

Additionally, an elif (short for «else if») clause can be added to the if statement to check for additional conditions:

```
if condition1:
    # execute code if condition1 is true
elif condition2:
    # execute code if condition1 is false and condition2 is true
else:
    # execute code if both condition1 and condition2 are false
```

Here, if condition1 is True, then the first block of code is executed. If condition1 is False, then condition2 is checked, and if it is True, then the second block of code is executed. If both condition1 and condition2 are False, then the last block of code is executed.

The correct syntax in Python for this statement is:

```
if 5 > 4:  
    print(True)  
else:  
    print(False)
```

This code will output «True» because the condition « $4 < 5$ » is true, and therefore the «if» statement is executed. This causes the «print(True)» statement to be executed.

Note that it's not necessary to include the «else» clause in this case, since there are only two possible outcomes and the condition is always true. So, the following code would have the same result:

```
if 5 > 4:  
    print(True)
```

This code will also output «True».

```
if 5 == 5:  
    print("yes")
```

This code will output «yes» because the condition «5 == 5» is true, and therefore the «print» statement is executed.

Note that in Python, the double equals sign «==» is used to check for equality between two values. In this case, we are checking if the value of 5 is equal to the value of 5, which is true.

```
x = 10  
y = 10  
if x != y:  
    print(True)  
else:  
    print(False)
```

In this case, the code will output «False» because the condition «x != y» is false. The values of both x and y are equal to 10, so the «if» statement is not executed, and therefore the «else» statement is executed instead. The «print(False)» statement is then executed, resulting in the output «False».

Note that the «else» statement is executed when the condition in the «if» statement is false. In this case, since x and y are both equal to 10, the condition is false, and the «else» statement is executed.

# Data Type

To print the type of the variable x, you need to put the print statement on a separate line:

```
x = 5  
print(type(x))
```

This will output the following:

```
<class 'int'>
```

This indicates that x is an integer, because the type() function returns the data type of the value that is passed to it.



# For Loops

In Python, the For loop statement is used to repeat the execution of a sentence or set of sentences repeatedly and by the number of elements in an array or string of characters.

The phrase for loop can be used in different forms, here are some examples:

The for loop is on the list:

The For loop statement can be used to execute repeated sentences on all items in a list. For example, if we have the following list:

```
fruits = ['apple', 'banana', 'cherry']
```

We can use the following for loop to print all fruits:

```
for fruit in fruits:  
    print(fruit)
```

And the fruits will be printed as follows:

```
apple  
banana  
cherry
```

The for loop on a string of characters:

The for loop can be used to execute repeated sentences on each character in the character string. For example, we can use the following for loop to print each character in the string «Python»:

The letters will be printed as follows:

```
P  
y  
t  
h  
o  
n
```

The for loop with the use of range:

The for loop can be used with the use of the range() function to execute repeated sentences a certain number of times. For example, we can use the following for loop to print numbers from 1 to 5:

```
for i in range(1, 6):  
    print(i)
```

The numbers will be printed as follows:

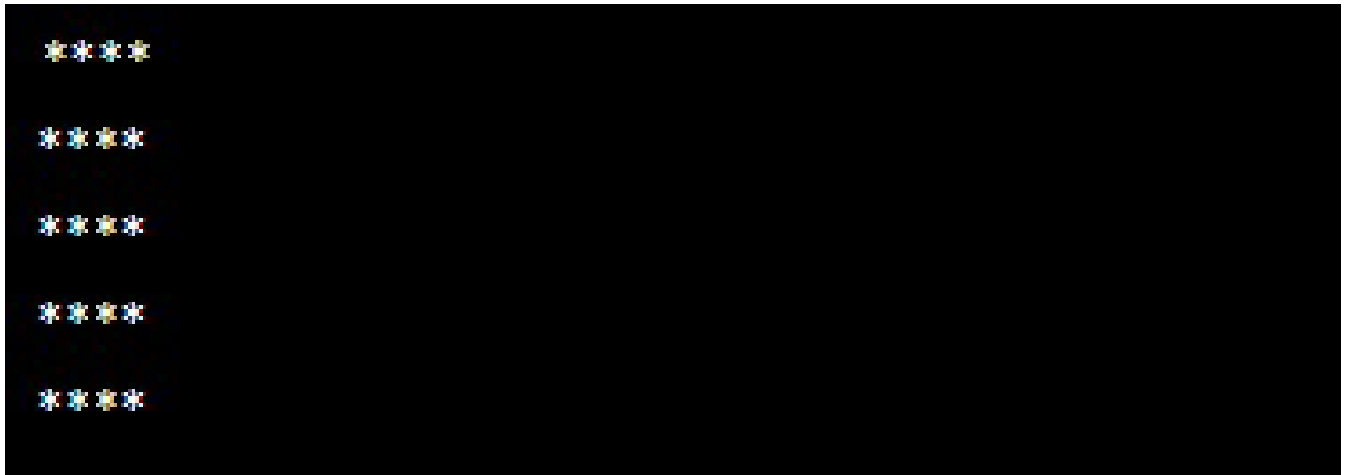
```
1  
2  
3  
4  
5
```

These are some examples of how the For loop phrase is used in Python. The for loop can be used in many different applications, such as executing repetitive sentences on lists

```
for i in range (5):  
    for x in range(4):  
        print('*',end="")  
    print()
```

This code uses nested for loops to print a pattern of stars. The code consists of two for loops inside each other. The outer loop is repeated 5 times, and the inner loop is repeated 4 times. At each iteration of the inner loop, one star is printed without moving the cursor to the new line using `end=""`, then after the inner loop is completed the new line moves using `print()` into the outer loop. This process is repeated until 5 rows of vertical stars are printed.

Thus, the output will be printed as follows:



Where 4 stars are printed on each line, and 5 rows of stars are printed.

```
for i in range(1,5):  
    for x in range(1,i+1):  
        print("*",end="")  
    print()
```

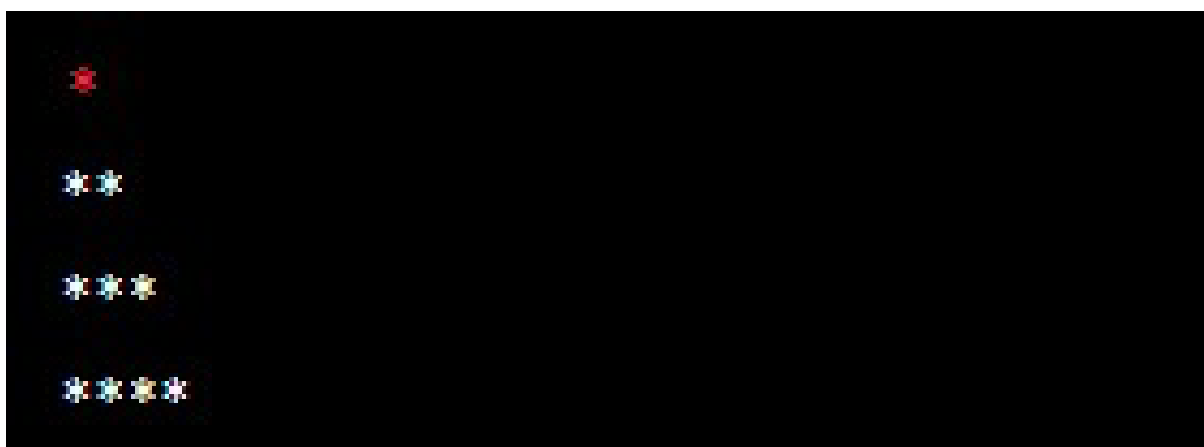
This code is used for loops to print a pattern of stars, where one line containing a star is printed and then more than one row containing stars is printed based on the number of lines. This is done using nested for-loops in this form:

The outer loop repeats 4 times using range (5 ,1) to repeat on four consecutive numbers 4 ,3 ,2 ,1.

The inner loop is duplicated based on the current value of the variable i (which is set in the outer loop) using range(1, i+1) to print stars by the current line.

At each cycle of the Inner Loop, a number of stars equal to the current value of the variable i is printed, and then the transition to the new line is carried out using print (), to print a new row.

Thus, the output will be printed as follows:



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

# Multiplication table

```
print("x*y|",end="")
for i in range(1,11):
    print(i,end="")
print()
print("-----",end="")
print()
for x in range(1,11):
    print(x,"|",end="")
    for y in range(1,11):
        print(x*y,end="")
    print()
    print(" ",end=" ")
```

When executed, the code you provided will generate a multiplication table from 1 to 10. The output of the program will look like this:

```
x*y|12345678910
-----
1|12345678910
2|2468101214161820
3|36912151821242730
4|481216202428323640
5|5101520253035404550
6|6121824303642485460
7|7142128354249566370
8|8162432404856647280
9|9182736455463728190
10|102030405060708090100
```



The first line of output displays the header row, which consists of the values from 1 to 10. The second line is a separator made up of a row of dashes. After that, the program uses nested for loops to generate the multiplication table. Each row of the table displays the products of multiplying the number on the left side (from 1 to 10) with the numbers at the top (also from 1 to 10).

For example, the third row shows the products of 3 multiplied by each number from 1 to 10. The product of 3 multiplied by 1 is 3, the product of 3 multiplied by 2 is 6, the product of 3 multiplied by 3 is 9, and so on, until the product of 3 multiplied by 10 is 30.

Overall, the output is a table that shows the products of multiplying each pair of numbers from 1 to 10.

# While Loops

While loops in Python are used for executing a block of code repeatedly until a certain condition is met. The syntax for a while loop in Python is:

```
while condition:  
    # code to be executed
```

The condition is a boolean expression that is evaluated before each iteration of the loop. If the condition is True, the code inside the loop is executed. Once the code inside the loop has been executed, the condition is evaluated again, and if it is still True, the code inside the loop is executed again. This process continues until the condition is False, at which point the loop is exited, and control is transferred to the next line of code after the loop.

Here is an example of a while loop in Python:

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

In this example, the variable *i* is initially set to 1. The while loop will continue to execute as long as the value of *i* is less than or equal to 5. Inside the loop, the value of *i* is printed, and then *i* is incremented by 1. This process will repeat until the value of *i* is 6, at which point the condition of the while loop will be False, and the loop will exit. The output of the above code will be:

```
1
2
3
4
5
```

This is a basic password verification program written in Python. The program sets a password variable to «5.1» and allows the user to input their guess for the password. If the guess does not match the password, the user is given 2 more chances to input the correct password. If the guess matches the password within the allowed number of attempts, the program outputs «valid password». If the guess is incorrect and the number of attempts is exceeded, the program outputs «try again later».

Here is how the program works:

```
Password=5.1
x=3
y=float(input("enter the password: ".title()))
x-=1
if Password !=y:
    print("you have",x,"chances")
while Password !=y and x>0:
    y = float(input("please try again :".title()))
    x-=1
if Password==y:
    print("valid password".title())
else:
    print("try again later".title())
```

Note that the program converts the input to a float using the float() function. If the input is not a valid float, a ValueError will be raised. Also, the program uses the title() method to capitalize the first letter of each word in the strings outputted by the program.

# List

A list in Python is a set of ordered and modifiable elements, inside of which elements can be stored using square brackets «[ ]» and separated by a comma»,». Any type of element can be stored inside a List, including numbers, text strings, other lists, and even functions.

When a List is created in Python, a space is allocated in memory to store all the elements added to it. The List can be modified by adding, removing or modifying elements within it. List items can also be accessed using indexing, since the position of the item in the List is determined using the index number.

An example of creating a List in Python:

```
my_list = [1, 2, 3, "four", "five"]
```

Items within a List can be accessed using indexing, for example:

```
print(my_list[0])  
print(my_list[3])
```

A new item can be added to the List using the append () command:

```
my_list.append("six")
```

An item can be deleted from the List using the remove () command:

```
my_list.remove(3)
```

The `pop()` function in Python is used to remove the element located at the specified location in the List and return its value. If the item location is not specified, the last item in the List will be removed and returned.

For example, if we have the following List:

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

And we want to remove the element at the specified location 2, we will use the `pop()` function as follows:

```
popped_item = my_list.pop(2)  
print(popped_item)
```



The `clear()` function in Python is used to remove all items in the List, and completely unpack them.

For example, if we have the following List:

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

And we want to remove all the elements in the List, we will use the `clear()` function as follows:

```
my_list.clear()  
print(my_list)
```

The result will be as follows:

```
[]
```

That is, the List now does not contain any elements.

# Tuple

A Tuple in Python is a sequence of Comma-separated elements enclosed in parentheses (), a data structure similar to a List but with important differences.

The main difference between Tuple and List is that Tuple cannot be changed after its creation, that is, it does not support mutable operations such as adding, deleting or modifying. Thus, Tuple is mainly used to store fixed sets of elements that are not changed later.

Tuple elements can be accessed using indexing, for example:

```
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple[0])
```

This command will produce one element of Tuple, where the value 1 that is located in the first position is printed in Tuple.

An empty Tuple can be created by giving an empty value to (), for example:

```
my_empty_tuple = ()
```

A Tuple can be created with a number of comma-delimited elements inside (), for example:

```
my_tuple = (1, 2, 3, 4, 5)
```

A Tuple can be created and its value assigned to a number of comma-delimited elements without the need to use (), for example:

```
my_tuple = 1, 2, 3, 4, 5
```

# Dictionary

A Dictionary in Python is a data structure that allows a programmer to store and retrieve values by keys instead of the usual indexing that is used in a List or Tuple.

A Dictionary in Python consists of a set of Comma-Separated Values surrounded by {} brackets, where each value in the Dictionary contains a key and a value (value) corresponding to this key.

The key can be any non-modifiable (immutable) value such as numbers and strings, while the value can be any value used in Python.

The values in the Dictionary can be accessed using the key, for example:

```
my_dict = {'apple': 1, 'orange': 2, 'banana': 3}  
print(my_dict['apple'])
```

This command will produce the value 1, where the <apple> key was used to access the specified key value.

An empty Dictionary can be created by giving an empty value to {}, for example:

```
my_empty_dict = {}
```

A Dictionary can be created with a set of specified values using parentheses {}, for example:

```
my_dict = {'apple': 1, 'orange': 2, 'banana': 3}
```

And new values can be added to the Dictionary using the specified key and their value, for example:

```
my_dict = {'apple': 1, 'orange': 2, 'banana': 3}  
my_dict['kiwi'] = 4
```

# Def

«def» is a reserved word (keyword) in the Python language used to define functions and methods in programming. The word «def» is used to specify the beginning of the definition of a function, followed by the name of the function, the list of expected inputs of the function (if the function expects inputs), after which the merge symbol»:», and followed by the function body that determines the behavior of the function when it is run.

An example of defining a function using «def» in Python:

```
def greet(name):  
    print("Hello, " + name + "!")
```

In this example, the reserved word «def» was used to define a new function as «greet» and specify a single input with the name «name». The function body contains an instruction that outputs a greeting message to the selected person using the input name.

This function can be called anywhere in the program by the function name, presenting the required data as operands, as follows:

```
greet("John")
```

This example will result from running the function and printing the text «Hello, John!»In directing.

Here are some examples of how to use «def» to define functions in Python:

A simple function for adding two numbers:

```
def add_numbers(a, b):  
    return a + b
```

A function for calculating the average of a list of numbers:

```
def calculate_average(numbers):  
    total = sum(numbers)  
    return total / len(numbers)
```

Function for printing custom greeting text:

```
def greet(name):  
    print("Hello, " + name + "!")
```



A function for converting temperature from the Fahrenheit scale to the Celsius scale:

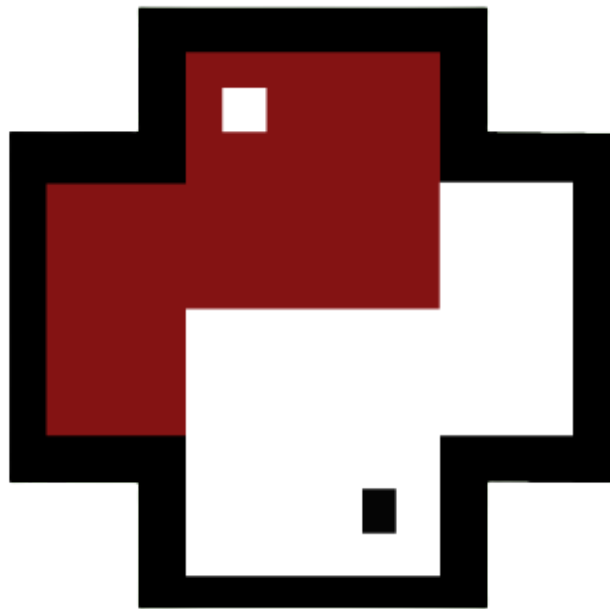
```
def fahrenheit_to_celsius(fahrenheit):  
    celsius = (fahrenheit - 32) * 5/9  
    return celsius
```

A function for converting a text string to an integer:

```
def string_to_integer(string):  
    return int(string)
```

«Def» in Python allows programmers to define custom functions and methods for performing certain functions in a program. These functions can be used to reuse code repeatedly and reduce the frequency of manual operations.

Good



Luck