* **Why Python?**

Python is the easiest, popular, and powerful programming language out there. We can use Python as a great tool to solve real-world problems.

I suggest Python for beginners because it is easy to learn. It is a high-level programming language with applications in numerous areas, including web development, machine learning, and many more.

* **Printing In Python**

We can print something in Python using the print statement.

Let’s see an example.

print("hello world")

This code will print the string “hello world” as the output.

If we want to print a numeric value or a variable, we will write it inside parenthesis without using quotes. I will tell you what a variable is in a moment. Now, let’s see an example of printing a numeric value.

print(1500)

This code will print the number 1500.

## Variables

We use variables in Python to store values. We can assign the variable a name

Let’s see an example code in which we assign the variable some value and use it.

applesCount = 50

print(applesCount)

This program will print 50 as the output.

## Getting User Input

Getting input from the user is an important task in programming. We can use the user input to do the required operations and produce the output.

In Python, user input is obtained by using the **input()** method. You can store the user input in a variable and use it for required operations.

Let’s see an example.

number = input()

We can also give instructions to the user for typing in the required input. These instructions can be written inside parentheses as an argument of the input() method.

Let’s see an example of that.

backgroundColor = input(" Enter a color: ")

This program will ask the user to enter the name of a color. The user can enter the name of a color, and it will be stored in the variable as a string

## Writing Comments

#This is a single line comment

Multiple line comments are created by placing the comment between  ”’  and  ”’.

Let’s see an example of multiple line comments in Python.

''' This is

a multiple-line

comment '''

You can use comments to enhance the readability of your code.

## Operators in Python

Operators are predefined symbols that can carry out particular operations. The participants of the operation are called operands.

There are many different types of operators available in Python. We can generally categorize them into the following types:

1. Arithmetic Operators
2. Increment and Decrement Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Assignment Operators

Now let’s discuss each category in detail.

### Arithmetic Operators

Arithmetic operators are those symbols that can do arithmetic calculations. In general, there are five of them.

|  |  |  |
| --- | --- | --- |
| **+** | Addition operator | We use the addition operator to carry out the addition of two operands. |
| **–** | Subtraction operator | The subtractor operator helps to carry out the subtraction of two numbers. |
| **\*** | Multiplication operator | We use it to multiply two operands. |
| **/** | Division operator | It helps to divide two operands. |
| **%** | Modulus operator | This operator gives the reminder value after the division of two operands. |

There are also two different types of arithmetic operators, unary and binary. Unary operators operate on a single operand. Operators applied to two operands are called binary operators.

### Increment and Decrement Operators

The increment operator (++) helps to increment a number, that is, adding 1 to the number. Also, the decrement operator (—) helps to decrement a number, that is, subtracting 1 from a number.

### Relational Operators

We can use relational operators for comparing two numbers to find the relation between them. The relational operators that are available in Python are:

|  |  |
| --- | --- |
| **==** | This operator checks whether the operand values are equal |
| != | This operator is used to check whether operand values are not equal |
| > | This operator checks whether the first operand is greater than the second one |
| < | This operand checks whether the first operand is lesser than the second one |
| >= | This operator checks whether the first operand is greater than or equal to the second one |
| <= | This operator checks whether the first operand is lesser than or equal to the second one |

### Logical Operators

We can use logical operators to obtain true or false values. There are mainly 3 logical operators:

|  |  |
| --- | --- |
| and | This operator returns true if both the operands on it’s either side are true. |
| or | This operator returns true if any of the operands on it’s either side are true. |
| not | This operator is used to reverse the logical state of an operand. |

### Bitwise Operators

Bitwise operators in Python are used to do operations on binary data.

* &    (Binary AND)
* |     (Binary OR)
* ^     (Binary XOR)
* ~      (Binary Ones Complement)
* <<    (Binary Left Shift)
* >>    (Binary Right Shift)

### Assignment Operator

= is used as the assignment operator in Python. This operator assigns a value to a variable.

We can also use a compound assignment operator like += to do addition and assignment within a single line of code.

For example,   a += 5  is equivalent to  a = a + 5.

## Expressions in Python

An expression in Python consists of operators and operands. The operands may be either constants or variables.

According to the type of result, we can classify expressions into arithmetic expressions and logical expressions.

## Arithmetic Expressions

If an expression contains only arithmetic operators, it is called an arithmetic expression. The result of an arithmetic expression will always be a numeric value. For example,  a + b.

They are of two types:

* Integer expressions
* Real expressions (floating-point expressions)

## Logical Expressions

If the result of the expression is either logic 1 (true) or logic 0 (false), then the expression is called logical expression. A logical expression may contain constants, variables, logical operators, and relational operators.

For example,  a <=  b

## Conditional Statements in Python

The three [conditional statements used in Python](https://pythonistaplanet.com/examples-of-conditional-statements-in-python/) are if, elif, and else. We can use these statements wisely to perform many tasks that involve some conditional checking.

These are very important to build logic in our programs. So, I recommend you give extra attention to conditional statements, loops, and functions while learning Python basics.

### if statement

We use ***if*** statements to determine what the program should do next after the statement is evaluated.

### else statement

The **else**statement runs if the **if**statement is false.

### elif statement

The **elif**statement allows you to include multiple **if**statements into one code block to handle various cases.

Let’s have a look at the syntax of these conditional statements.

if condition :

statements

elif condition:

statements

else:

statements

### Indentation in Python

Python uses indentation (white space at the beginning of a line) to delimit blocks of code. Other languages such as C, Java, C++, etc., use curly braces { } to accomplish this, but in Python, indentation is mandatory.

### Loops in Python

It is necessary to understand the working of loops in order to develop your programming skills. Loops allow us to execute a statement or group of statements multiple times. Thus we can reduce the burden of repeatedly writing a piece of code many times.

### while loop

The while loop checks a condition and executes the task if the condition is satisfied.   The syntax of the while loop looks like this:

while condition:

statements

Let’s look at an example.

count=1

while count<=5:

print(count)

count = count+1

The output of this code will be 1 2 3 4 5.

We use the **break**statement to end a loop prematurely.

We can use the **continue**statement to stop the current iteration and continue with the next one.

### for loop

Iterating through a list using a while loop requires quite a lot of code. That’s why Python provides the **for loop**.

Let’s see an example:

words = ['hello','world','hi']

for word in words:

print(word)

This code prints all three words separately as output.

The for loop is commonly used to repeat some code a certain number of times. We can do this by combining “for loops” with range objects.

For example :

for i in range(3):

print('hello')

This code prints hello three times as output.

### Range

The range function creates a sequential list of numbers. Let’s see an example.

numbers = list(range(10))

print(numbers)

This will produce an output like: [0,1,2,3,4,5,6,7,8,9].

We can use two arguments to set the starting and ending points.

For example:

numbers = list(range(3,10))

print(numbers)

This will give the output like: [3,4,5,6,7,8,9]

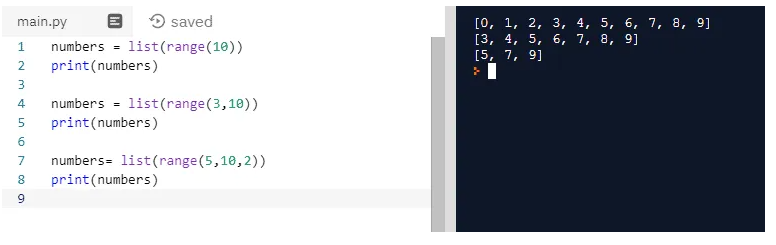
The range method can have a third argument, which determines the interval of the sequence produced. The argument must an integer.

For example:

numbers= list(range(5,10,2))

print(numbers)

The output of this code will be  [5,7,9].



## Functions in Python

Any statement that consists of a word followed by information in parenthesis is a function call.

For example:

print('hello')

In addition to the predefined functions, we can create our own functions by using the def statement.

Let’s see an example.

def greetings():

print('Hey')

print('Hellooooo')

greetings()

The first three lines are the function definition, and the last line is the function call. You can call the function whenever you want, once it is defined.

When we call this method, the output will be:

‘Hey’ ‘Hellooooo?’

The statements in the function are executed only when they are called. We must define our functions before we call them.

Functions can take arguments (one or more).

Let’s see an example function to add two numbers and print their sum.

def add(x,y):

print(x+y)

add(5,8)

This function call will produce an output: 13.

Certain functions return a value that can be used later. To do this, we can use the return statement.

Let’s see an example function to find the largest of two numbers.

def max(x,y):

if x>=y:

return x

else:

return y

z=max(8,5)

print(z)

This code will produce an output 8.

When a function finds a return statement, it immediately stops its execution. The function will not consider the code that we write after the return statement.

There are different types of function arguments in Python. If you want to learn more about those, check out [this article](https://pythonistaplanet.com/types-of-function-arguments-in-python/).

## Strings in Python

A string is created by entering a text between two single or double quotes.

For example, “Pythonista Planet” and ‘Learn Python Basics’ are strings.

### String Concatenation

We can join together two strings using the + operator.

For example, ‘Pythonista’ + ‘Planet’ gives ‘Pythonista Planet’.

We can also multiply strings in Python.

For example, ‘mango’ \* 3  gives  ‘mangomangomango’.

To get the letters of a string separately, we can use the index of the letter inside square brackets.

For example,  ‘python'[0]    gives   ‘p’.

### String Slicing

To slice a string, use the starting and ending index inside square brackets (like [ start : stop ]  ).

For example,   ‘python'[0:2]   gives   ‘py’ .

And  ‘python'[0:4]  gives  ‘pyth’.

If we want to slice the string from the other end, we can do that by using negative numbers.

For example, ‘python'[-1]   gives  ‘n’ .

Also,  ‘python'[0:-1]  gives  ‘pytho’.

We can also slice the string by doing stepping.    [ start : stop : step ]

For example,   ‘python'[0:-1:2]   gives   ‘pto’.

Also,   ‘howareyou'[0:-1:3]  gives  ‘hay’.

### Splitting a String

To split a sentence into a list of words, we can use the split() method.

sentence = "Pythonista Planet"

words = sentence.split()

print(words)

This will produce the output: [‘Pythonista’,’Planet’].

## Lists in Python

If we want to store a set of items in an arranged order, we can use lists in Python. We can create a list by using square brackets with items inside. We can use commas to separate each item.

If we want to access a particular item in a lost, we can do that by using the index of that item in square brackets.

For example:

words= ['Hello','world','how','are','you']

print(words[0])

This will produce an output ‘Hello‘.

Often, a list contains items of the same type. But, we can also create a list that includes different types of items as well. Lists can also be nested within other lists.

Let’s see an example.

things= ['mango',0,[1,2],4.56]

print(things[2][1])

This will produce the output: 2.

Strings can also be indexed like lists.

For example:

message="hello world"

print(message[6])

This code will produce an output ‘w’.

## List functions

Now, let’s look at some of the useful methods available in [Python for working with lists](https://pythonistaplanet.com/23-examples-of-using-lists-in-python/).

### append

The append method adds an item to the end of an existing list.

For example:

numbers= [1,2,3]

numbers.append(4)

print(numbers)

This code will produce the output: [1,2,3,4].

### insert

The insert method allows inserting a new item to the end of an existing list.

For example:

words = ['python','fun']

words.insert(1,'is') # (position,item)

print(words)

The output of this code will be: [‘python’,’is’,’fun’].

### len

If we want to get the number of items in a list, we can use the **len** method.

Let’s see an example.

numbers= [1,2,3,4,5]

print(len(numbers))

The output will be 5.

There are a few more useful functions and methods for lists like :

|  |  |
| --- | --- |
| max(list) | To find the maximum value from a list |
| min(list) | To find the minimum value from a list |
| list.remove(item) | To remove an item from a list |
| list.reverse() | To reverse a list |

## Modules in Python

Some great developers have written a lot of code to simplify common tasks. Those pieces of code are called modules. These are also called libraries.

Just think of it like Tony Stark designed a powerful Iron Man suit for you, and you can just wear it to have some superpower.

Modules are kind of like this. Some genius people have already spent their sleepless nights writing code in order to make your life easier.

You can just import the modules and call some methods to do the tasks. Problem solved in just two or three lines of code.

Let’s see how we can use the superpower of modules.

You can use a Python module with the help of the import statement. Include ‘import module\_name‘ at the beginning of the code. Then you can use the functions and values available in the module using the dot operator.

Consider an example of using the ‘random’ module.

import random

for i in range(5):

value= random.randint(90,99)

print(value)

The output of this code will be some other random values between 90 and 99.

If you need only certain methods or values from a module, you can use the format: ‘from module\_name import function\_name’.

For example :

from math import sqrt,pi

By using the ‘as’ statement, we can import a module under a different name. This can be used when a module or function has a long or confusing name.

For example :

from math import sqrt as square\_root

Trying to import a module that is not available will cause ImportError.

There are three main types of modules in Python: those you write yourself, those you install from external sources, and those that are pre-installed with Python.

### Standard Library in Python

The modules that are pre-installed with Python constitute the standard library. Some of the standard library’s useful modules are string, turtle, math, random, datetime, os, multiprocessing, socket, email, json, doctest, etc.

Python’s standard library is one of its main strengths as a programming language.

## Dictionaries in Python

[Dictionaries](https://pythonistaplanet.com/examples-of-dictionaries-in-python/) are a special type of data structure in python. They are used to store key-value pairs. So if you put in a key, it gives back the value.

The syntax of dictionaries would look like this:

{ key1: value1, key2 : value,…. }  
  
For example, if we want to create a phonebook, then we will create a dictionary like this.

phone\_book = {

'John' : '8592000000',

'Bob' : '7994000000',

'Elisa' : '9749777777'

}

After that, if we go to the contacts and type in the name, it will give you the phone number.

print(phone\_book['John'])

Then, the output will be: ‘8592000000’

What if we want to store more values along with phone numbers, say we want to add email too. Then, we use lists inside dictionaries.

phone\_book = {

'John' : ['8592000000', 'john@xyzmail.com'],

'Bob' : ['7994000000', 'bob@xyzmail.com'],

'Elisa' : [ '9749777777' , 'elisa@xyzmail.com']

}

print(phone\_book['John'])

The output of this code will be: ‘8592000000’, ‘john@xyzmail.com’

What if we want only the phone number of John from this dictionary. Then we use the index of the list.

Let’s look at an example:

print(phone\_book['John'][0])

The output will be: ‘8592000000’

### Adding a new item into the dictionary

Let’s see how we can add a new key-value to this dictionary. We can do this by writing the following line of code.

phone\_book[ 'Adam' ] = '9567974800'

Now, imagine that one person in your phonebook changed his phone number to another one. So, in that case, we want to update our dictionary, right? Let’s see how we can do this.

phone\_book[ 'John' ] = '1567933400'

Now, John’s phone number is changed to ‘1567933400’.

### Looping through a dictionary

We can loop through a dictionary using the following syntax.

for k,v in phone\_book.items():

print(k, ":", v)

Here, k and v represent key and value respectively.

## Tuples in Python

Tuples are very similar to lists, except that they are immutable, that is, they can’t be changed once defined. Also, they are represented using parentheses rather than square brackets.

Let’s look at an example of a tuple.

exam\_register\_numbers = ('15001147' , '15001148' )

If you want to access the values in the tuple, you can do that using the corresponding indices.

For example :

print(exam\_register\_numbers[0])

The output will be: ‘15001147’.

Trying to change a value in the tuple causes TypeError since they are immutable. Tuples can be nested within each other, just like lists. Tuples are faster than lists.

## Mutable vs Immutable Data Types

Mutable data types are those that can be changed. For example, lists are mutable.

Let’s see an example :

countries = [ 'India' , 'Poland' , 'Germany' ]

countries[2] = 'U.S.A'

print(countries)

The output for this code will be  ‘India’, ‘Poland’, ‘U.S.A’

Immutable data types are those that can’t be changed. For example, tuples are immutable.

Let’s look at the example:

ucl\_semifinal = ('Real\_Madrid', 'Liverpool' , 'Bayern\_Munich' , 'Roma' )

ucl\_semifinal[3] = 'Barcelona'

print(ucl\_semifinal)

In this case, the second line shows a TypeError.

If you are a Barcelona football club fan, don’t take it seriously. I just wrote this code to show you an example.

## Sets in Python

A set is an unordered collection of items. All the elements in a set are unique and immutable. However, the set itself is mutable. We can insert or delete items from it if we want.

A set is created by placing all the items inside curly braces, and the items are separated by a comma.

For example:

groceries = { 'apple', 'milk', 'chicken'}

A set can have different types of items. We can’t access or change an element of a set using indexing or slicing. Sets in Python do not support it.

We can add a single element using add() method and multiple elements using the update() method. A particular item can be removed from the set using methods, discard() and remove().

We can remove and return an item using the pop() method (arbitrary item). If we want to remove all items from a set, we can use the **clear**method for doing that.

groceries = {'apple', 'mango', 'apple'}

print(groceries)

The output will be :  {‘apple’,’mango’} , since every element is unique.

Sets can also perform mathematical set operations like union, intersection, symmetric difference, etc.

For example :

A = {2,3}

B = {1,2}

print(A|B)

print(A&B)

print(A-B)

The output of this program will be :  
                                         {1,2,3}  
                                         {2}  
                                         {3}

## String Formatting in Python

Now, let’s move to the next level of Python basics. A little more advanced concepts like [string formatting](https://pythonistaplanet.com/python-format-function/) and exception handling.

String formatting provides a powerful way to combine non-strings within strings. It uses the ***format()*** method to substitute a number of arguments in the string.

Let’s see an example.

fruits=['apple','orange','mango']

bucket = "The fruits inside the bucket are : {0} , {1} and {2} .".format( fruits[0],fruits[1],fruits[2] )

print(bucket)

The output of this piece of code will be: ‘The fruits inside the bucket are apple, orange, and mango’.

Each argument of the format() method is placed in the string at the corresponding positions, which is determined using the curly braces {}.

We can also do string formatting with named arguments.

Let’s see an example.

goals= "Ronaldo : {rono} , Messi : {leo} ".format(rono=28,leo=27)

print(goals)

The output will be: ‘Ronaldo: 28, Messi: 27’

Let’s see one more example case.

print("{0}{1}{0}".format("abra","cad"))

The output for this code is ‘abracadabra’.

## File Handling in Python

We can handle files using Python and perform operations like opening, reading, and writing on the files. Let’s see how we can open files, read the contents of the files, and write to the files using Python.

There are some modes available in Python to handle the files. They are:

* “r” – read mode (for reading the contents of the file)
* “w” – write mode (for writing content to the file)
* “a” – append mode (for adding new content to the end of the file)
* “b” – binary mode (for non-text files such as images and audio files)

### Opening Files

To open files using Python, we can use the open() function and pass the file path and mode as arguments.

Syntax: variable = open(“file path”, “mode”)

Whenever we open a file, it must be closed after use. We can close files using the close() method.

Let’s see an example. I’ve created a text file called “myfile.txt”. Let’s see how we can open it.

x = open("myfile.txt","r")

x.close()

### Reading Files

We can read the contents of a file using Python with the help of the read() method. Let’s see an example.

I’ve written a sentence “Hello World” in the “myfile.txt” file.

Now, let’s see how we can read this file using Python and print the contents on the console.

x = open("myfile.txt","r")

content = x.read()

print(content)

x.close()

When we run this code, we will be able to see the file contents as output.

In the read() method, you can specify a number if you want to read only a certain number of bytes.

x = open("myfile.txt","r")

content = x.read(5)

print(content)

x.close()

You can also use the readlines() method to return a list in which each element is a line in the file.

x = open("myfile.txt","r")

content = x.readlines()

print(content)

x.close()

### Writing to Files

We can write to files by opening the files “w” mode and using the write() method to write the contents. Let’s see an example.

x = open("myfile.txt","w")

x.write("A Python coder has hacked this file!")

x.close()

Now, if we open the file, we will see that the file content has been changed.

The “w” mode will create a new file if the file doesn’t exist. When a file in the system is opened using “w” mode, the existing content in the file will be deleted.

## Exception Handling in Python

We can handle exceptions in Python using try-except statements. An exception is an event that occurs due to incorrect code or input. When an exception occurs, the program immediately stops. Exception handling is useful when dealing with user inputs.

The different types of exceptions are:

|  |  |
| --- | --- |
| ZeroDivisionError | It occurs when we divide something by 0. |
| ImportError | It occurs when an import fails. |
| IndexError | It occurs when a list is indexed with an out-of-range number. |
| NameError | It occurs when we use an unknown variable. |
| SyntaxError | It occurs when the code can’t be parsed properly. |
| TypeError | It occurs when we call a function on a value of an inappropriate type. |
| ValueError | It occurs when we call a function on a value of the correct type, but with an inappropriate value. |

### try-except

To handle exceptions and to call some code when an exception occurs, you can use a try/except statement. The try block contains the code that we want to test.

If that exception occurs, the code in the try block stops its execution, and the code in the except block runs. The code in the except block will not run if there are no errors.

try:

num1 = 7

num2 = 0

print(num1/num2)

print("Done calculation")

except ZeroDivisionError:

print("Error! Do not divide by zero")

The output will be: ‘Error! Do not divide by zero‘.

We can also use multiple except blocks at the same time to handle different exceptions.

Let’s see an example:

try:

mango=10

print(mango+" juice ")

print(mango/2)

except ZeroDivisionError:

print(" Division by zero. ")

except (ValueError,TypeError):

print("Error")

The output will be: ‘Error‘

We can simply put an except statement without specifying any exceptions. It will catch all the errors. This should be used sparingly, as they can catch unexpected errors and may hide programming mistakes.

Let’s see an example :

try:

mango = " juice "

print(mango/0)

except:

print("Error")

The output will be: ‘ Error ‘

### finally

We can use the**finally** statement at the end of a try/except block to ensure that some code will run even if there are errors.

try:

print("Hello")

print(1/0)

except ZeroDivisionError:

print(" Division by zero ")

finally:

print("This code will run always")

The output of this python code will be:

‘ Hello ‘

‘Division by zero’

‘ This code will run always ‘

## Conclusion

It was a long tutorial. A thumbs up to you if you have followed it completely till the end.

I hope this article was helpful. You can bookmark this article to use it as a reference. Whenever you have any doubts, you can come back to them and clear your doubts.

If you want to practice what you’ve learned, I’ve written an article on [35 Python Programming Exercises and Solutions](https://pythonistaplanet.com/python-programming-exercises-and-solutions/). Check it out and practice coding.

Now you know the basics of Python. It is time to learn more advanced skills. I would highly recommend you learn some object-oriented programming concepts as well. That will help you whenever you make software applications. You can [check this article](https://pythonistaplanet.com/python-oops-concepts/) to learn object-oriented concepts in Python.

After that, you can learn web development, machine learning, [desktop app](https://pythonistaplanet.com/create-apps/) development, etc. Python has a lot of applications.

If you have any doubts, feel free to comment down below. I’ll be happy to help you.

I spent a long time making this article. I would appreciate it if you would be willing to share it. It will inspire me to create more useful tutorials like this.

Happy coding.