**juPython Script**

In computing, the word script is used to refer to a file containing a logical sequence of orders or a batch processing file. This is usually a simple program, stored in a plain text file. Scripts are always processed by some kind of interpreter, which is responsible for executing each command sequentially.

A plain text file containing Python code that is intended to be directly executed by the user is usually called **script. Steps of Script Execution:**

1. Process the statements of your script in a sequential fashion
2. Compile the source code to an intermediate format known as bytecode

This bytecode is a translation of the code into a lower-level language that’s platform-independent. Its purpose is to optimize code execution. So, the next time the interpreter runs your code, it’ll bypass this compilation step.

Strictly speaking, this code optimization is only for modules (imported files), not for executable scripts.

1. Ship off the code for execution

At this point, something known as a Python Virtual Machine (PVM) comes into action. The PVM is the runtime engine of Python. It is a cycle that iterates over the instructions of your bytecode to run them one by one.

The PVM is not an isolated component of Python. It’s just part of the Python system you’ve installed on your machine. Technically, the PVM is the last step of what is called the Python interpreter.

The whole process to run Python scripts is known as the Python Execution Model.

How to write python scripts

Command line Interpreter

The commands can be directly entered onto the interpreter.

Python also works in interactive mode, which allows you to enter Python commands using a command-line interface. This is a great resource for immediate Python gratification. You need to locate and start up your computer’s command-line interface, sometimes known as the terminal or terminal emulator, or on machines running Windows, the DOS prompt. This will provide you with a command line at which you can start typing commands. Interactive mode can be started up by typing python at the command line. The details may be different according to your operating system. The first line shows the Python version number and the compilation date of the version you have installed. The last line (>>>) is your Python prompt. This indicates that the Python interpreter is running and waiting for your input. The Python interpreter is the program that reads Python programs and carries out their instructions. Interactive mode can be useful for testing out individual commands line by line, experimenting with new ideas, and getting help

Using a text file

A Python interactive session will allow you to write a lot of lines of code, but once you close the session, you lose everything you’ve written. That’s why the usual way of writing Python programs is by using plain text files. By convention, those files will use the .py extension. (On Windows systems the extension can also be .pyw.). the file is also called a script file, batch file or an application. Python scripts can also be imported by other scripts. In that case they are known as modules.

pi=3.14

r=10

c=2\*pi\*r

print(c)

import hello

Python code files can be created with any plain text editor like notepad. Save it with .py extension . It can be run by double clicking on it

Using an IDE

An integrated development environment (IDE) is a graphical interface with lots of useful features designed to make programming with Python even easier. There are now several Python-specific IDEs. Popular applications include IDLE (which is bundled with Python itself), Eric, Geany, DrPython, and SPE.

Spyder is a free and open source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

Identifiers

Python identifiers are user-defined names. They are used to specify the names of variables, functions, class, module, etc.Rules to define python identifiers

* You can’t use reserved [keywords](https://www.askpython.com/python/python-keywords) as an identifier name.
* Python identifier can contain letters in a small case (a-z), upper case (A-Z), digits (0-9), and underscore (\_).
* Identifier name can’t begin with a digit.
* Python identifier can’t contain only digits.
* Python identifier name can start with an underscore.
* There is no limit on the length of the identifier name.
* Python identifier names are case sensitive.
* Valid Identifiers
  + ab10c: contains only letters and numbers
  + abc\_DE: contains all the valid characters
  + \_: surprisingly but Yes, underscore is a valid identifier
  + \_abc: identifier can start with an underscore

use string isidentifier() function to check if the identifier name is valid or not.

print("abc".isidentifier())  # True

print("99a".isidentifier())  # False

print("\_".isidentifier())  # True

print("for".isidentifier())  # True - wrong output

* Class names should start with capital letters. For example Person, Employee, etc.
* If the class name has multiple words, use Uppercase for the first character of each word. For example EmployeeData, StringUtils, etc.
* You should use small letters for variables, functions, and module names. For example, collections, foo(), etc.
* If variables, functions, and module names have multiple words then separate them with an underscore. For example, is\_empty(), employee\_object, etc.
* For private variables, you can start their names with an underscore.
* Avoid underscore as the first and last character in the identifier name. It’s used by python built-in types.
* If the identifier starts and ends with two underscores, then it means that the identifier is a language-defined special name. So you should avoid having two underscores at the start and the end of the identifier name.
* Keep identifier names meaningful to clarify their intent. For example, phone\_number, is\_uppercase, etc.
* If a function returns boolean value, it’s better to start its name with “is”. For example, isidentifier, iskeyword, etc.
* There is no limit on the length of the identifier name. But, keep it small and to the point. For example, the\_employee\_object\_first\_name can be better named as emp\_first\_name.

Keywords

 python keyword is a reserved word which you can’t use as a name of your variable, class, function etc. These keywords have a special meaning and they are used for special purposes in Python programming language. For example – Python keyword “while” is used for while loop thus you can’t name a variable with the name “while” else it may cause compilation error.

To look for available keywords, open help() and type keyword. It will return list of keywords.

False break for not

None class from or

True continue global pass

\_\_peg\_parser\_\_ def if raise

and del import return

as elif in try

assert else is while

async except lambda with

await finally nonlocal yield

Keyword.iskeyword(“for”) returns true if the argument is a keyword.

Data types:

Data types are the classification or categorization of data items.  This **data type defines** the operations that can be done on the **data**, the meaning of the **data**, and the way values of that **type** can be stored.It represents the kind of value that tells what operations can be performed on a particular data. In many programming languages, there are two stages to creating a variable: The first is to create the container and stick an identifying label on it; this is called initialization. The second is to put a value into it; this is called assignment. Initialization and assignment are performed with a single command in Python, using the = sign. A section of code that does something, such as an assignment, is known as a statement. The part of the code that can be evaluated to produce a value is known as an expression. The right-hand side of an assignment can be an expression, like the assignment to total\_price in the following list of simple assignment statements: number = 0 roll\_width = 1.4 price\_per\_metre = 5 filename = 'data.txt' trace = False sentence = "this is a whole lot of nothing" total\_price = roll\_width \* price\_per\_metre

In Python, once a value has been assigned to a variable, the interpreter will then decide what sort of value it is (i.e., a number, some text, or some other relevant quality). This is known as dynamic typing.Dynamic typing makes it much easier to handle different types of unpredictable user input. The interpreter can accept user input in many different forms, to which it assigns a type dynamically. This means that a single piece of code can be used to deal with words, numbers, or any other data type, and that the programmer doesn’t need to decide what type the data will be in order to assign it to a variable

Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. **id**() is a built-in **function in Python** 3, which **returns** the **identity** of an object. The **identity** is a unique integer for that object during its lifetime. This is also the **address** of the object in **memory**.



Numeric

In Python, numeric data type represent the data which has numeric value. They are immutable data types. This means, changing the value of a number data type results in a newly allocated object. Number objects are created when you assign a value to them. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

**Integers**– This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be. Integers can be binary, octal, and hexadecimal values. he numbers we deal with every day are of the decimal (base 10) number system. But computer programmers (generally embedded programmers) need to work with binary (base 2), hexadecimal (base 16) and octal (base 8) number systems.

In Python, we can represent these numbers by appropriately placing a prefix before that number. The following table lists these prefixes.

* The following strings can be prepended to an integer value to indicate a base other than 10:

| **Prefix** | **Interpretation** | **Base** |
| --- | --- | --- |
| 0b (zero + lowercase letter 'b') 0B (zero + uppercase letter 'B') | Binary | 2 |
| 0o (zero + lowercase letter 'o') 0O (zero + uppercase letter 'O') | Octal | 8 |
| 0x (zero + lowercase letter 'x') 0X (zero + uppercase letter 'X') | Hexadecimal | 16 |

>>> 0b11011000 # binary

216

>>> 0o12 # octal

10

>>> 0x12 # hexadecimal

15

* **Float**– This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.

>>> 4.2

4.2

>>> type(4.2)

<class 'float'>

>>> 4.

4.0

>>> .2

0.2

>>> .4e7

4000000.0

>>> type(.4e7)

<class 'float'>

>>> 4.2e-4

0.00042

Almost all platforms represent Python float values as 64-bit “double-precision” values, according to the [IEEE 754](https://en.wikipedia.org/wiki/IEEE_754_revision) standard. In that case, the maximum value a floating-point number can have is approximately 1.8 ⨉ 10308. Python will indicate a number greater than that by the string inf:

>>> 1.79e308

1.79e+308

>>> 1.8e308

inf

The closest a nonzero number can be to zero is approximately 5.0 ⨉ 10-324. Anything closer to zero than that is effectively zero:

>>> 5e-324

5e-324

>>> 1e-325

0.0

We can format integers as binary, hexadecimal, etc. and floats can be rounded or displayed in the exponent format.

>>> # formatting integers

>>> "Binary representation of {0} is {0:b}".format(12)

'Binary representation of 12 is 1100'

>>> # formatting floats

>>> "Exponent representation: {0:e}".format(1566.345)

'Exponent representation: 1.566345e+03'

>>> # round off

>>> "One third is: {0:.3f}".format(1/3)

'One third is: 0.333'

## Type Conversion

We can convert one type of number into another. This is also known as coercion.

Operations like addition, subtraction coerce integer to float implicitly (automatically), if one of the operands is float.

>>> 1 + 2.0

3.0

We can see above that 1 (integer) is coerced into 1.0 (float) for addition and the result is also a floating point number.

We can also use built-in functions like int(), float() and complex() to convert between types explicitly. These functions can even convert from [strings](https://www.programiz.com/python-programming/string).

>>> int(2.3)

2

>>> int(-2.8)

-2

>>> float(5)

5.0

>>> complex('3+5j')

(3+5j)

When converting from float to integer, the number gets truncated (decimal parts are removed).

## Python Decimal

Python built-in class float performs some calculations that might amaze us. We all know that the sum of 1.1 and 2.2 is 3.3, but Python seems to disagree.

>>> (1.1 + 2.2) == 3.3

False

What is going on?

It turns out that floating-point numbers are implemented in computer hardware as binary fractions as the computer only understands binary (0 and 1). Due to this reason, most of the decimal fractions we know, cannot be accurately stored in our computer.

Let's take an example. We cannot represent the fraction 1/3 as a decimal number. This will give 0.33333333... which is infinitely long, and we can only approximate it.

It turns out that the decimal fraction 0.1 will result in an infinitely long binary fraction of 0.000110011001100110011... and our computer only stores a finite number of it.

This will only approximate 0.1 but never be equal. Hence, it is the limitation of our computer hardware and not an error in Python.

>>> 1.1 + 2.2

3.3000000000000003

To overcome this issue, we can use the decimal module that comes with Python. While floating-point numbers have precision up to 15 decimal places, the decimal module has user-settable precision.

Let's see the difference:

import decimal

print(0.1)

print(decimal.Decimal(0.1))

**Output**

0.1

0.1000000000000000055511151231257827021181583404541015625

This module is used when we want to carry out decimal calculations as we learned in school.

It also preserves significance. We know 25.50 kg is more accurate than 25.5 kg as it has two significant decimal places compared to one.

from decimal import Decimal as D

print(D('1.1') + D('2.2'))

print(D('1.2') \* D('2.50'))

**Output**

3.3

3.000

Notice the trailing zeroes in the above example.

why not implement Decimal every time, instead of float? The main reason is efficiency. Floating point operations are carried out must faster than Decimal operations.

### When to use Decimal instead of float?

We generally use Decimal in the following cases.

* When we are making financial applications that need exact decimal representation.
* When we want to control the level of precision required.
* When we want to implement the notion of significant decimal places.

## Python Fractions

Python provides operations involving fractional numbers through its fractions module.

A fraction has a numerator and a denominator, both of which are integers. This module has support for rational number arithmetic.

We can create Fraction objects in various ways. Let's have a look at them.

import fractions

print(fractions.Fraction(1.5))

print(fractions.Fraction(5))

print(fractions.Fraction(1,3))

**Output**

3/2

5

1/3

While creating Fraction from float, we might get some unusual results. This is due to the imperfect binary floating point number representation as discussed in the previous section.

Fortunately, Fraction allows us to instantiate with string as well. This is the preferred option when using decimal numbers.

import fractions

# As float

# Output: 2476979795053773/2251799813685248

print(fractions.Fraction(1.1))

# As string

# Output: 11/10

print(fractions.Fraction('1.1'))

**Output**

2476979795053773/2251799813685248

11/10

This data type supports all basic operations. Here are a few examples.

from fractions import Fraction as Frequency

a=F (1,5)

b=

c=

d=a+b+c

print(F(1, 3) + F(1, 3))

print(1 / F(5, 6))

print(F(-3, 10) > 0)

print(F(-3, 10) < 0)

**Output**

2/3

6/5

False

True

**Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

You can delete a single object or multiple objects by using the **del** statement. For example −

del var

del var\_a, var\_b

z=

x = 1.0

y = 1.0

# converting x and y into complex number

z = complex(x,y);

**Extracting real and imagery parts of complex number**

Z=1+2j

z.real=1.0

z.imag=2.0

Both real and imagery parts are stored as type float. Check using type function

**Addition of two complex numbers**

**Input:** 2+3i, 4+5i

Sum=z1+z2

**Output:** Addition is: 6+8i

**Input:** 2+3i, 1+2i

**Output:** Addition is: 3+5i

Subtraction of two complex numbers

Diff =z1-z2

**Input:** 2+3i, 4+5i

**Output:** Subtraction is: -2-2i

**Input:** (2+3i)(1+2i)=2+3i+4i-6=-4+7i

**Output:** Subtraction is: 1+1i

Multiplication of two complex numbers

Product =z1\*z2

**String**

A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters. Computers do not deal with characters, they deal with numbers (binary). internally it is stored and manipulated as a combination of 0s and 1s.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encodings used. In Python, a string is a sequence of Unicode characters. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

# Python string examples - all assignments are identical.

String\_var = 'Python'

String\_var = "Python"

String\_var = """Python"""

# with Triple quotes Strings can extend to multiple lines

String\_var = """ This document will help you to

explore all the concepts

of Python Strings!!! """

# Replace "document" with "tutorial" and store in another variable

substr\_var = String\_var.replace("document", "tutorial")

print (substr\_var)

## How to access characters in a string?

We can access individual characters using indexing and a range of characters using slicing.  
 Index starts from 0. Trying to access a character out of index range will raise an IndexError. The index must be an integer. We can't use floats or other types, this will result into TypeError.

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| h | E | L | L | O | ! |
| 0 | 1 | 3 | 4 | 5 | 6 |
| -6 | -5 | -4 | -3 | -2 | -1 |

We can access a range of items in a string by using the slicing operator :(colon).

str = 'suchit purohit'

print('str = ', str)

#first character

print('str[0] = ', str[0])

#last character

print('str[-1] = ', str[-1])

#slicing 2nd to 5th character

print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character

print('str[5:-2] = ', str[5:-2])

### Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The **+** operator does this in Python. Simply writing two string literals together also concatenates them.

The **\*** operator can be used to repeat the string for a given number of times.

# Python String Operations

str1 = 'Hello'

str2 ='World!'

# using +

print('str1 + str2 = ', str1 + str2)

# using \*

print('str1 \* 3 =', str1 \* 3)

### String Membership Test

We can test if a substring exists within a string or not, using the keyword in.

>>> 'a' in 'program'

True

>>> 'at' not in 'battle'

False

## Python String Formatting

### Escape Sequence

If we want to print a text like He said, "What's there?", we can neither use single quotes nor double quotes. This will result in a SyntaxError as the text itself contains both single and double quotes.

>>> print("He said, "What's there?"")

...

SyntaxError: invalid syntax

>>> print('He said, "What's there?"')

...

SyntaxError: invalid syntax

One way to get around this problem is to use triple quotes. Alternatively, we can use escape sequences.

An escape sequence starts with a backslash and is interpreted differently. If we use a single quote to represent a string, all the single quotes inside the string must be escaped. Similar is the case with double quotes. Here is how it can be done to represent the above text.

# using triple quotes

print('''He said, "What's there?"''')

# escaping single quotes

print('He said, "What\'s there?"')

# escaping double quotes

print("He said, \"What's there?\"")

Here is a list of all the escape sequences supported by Python.

|  |  |
| --- | --- |
| Escape Sequence | Description |
| \newline | Backslash and newline ignored |
| \\ | Backslash |
| \' | Single quote |
| \" | Double quote |
| \a | ASCII Bell |
| \b | ASCII Backspace |
| \f | ASCII Formfeed |
| \n | ASCII Linefeed |
| \r | ASCII Carriage Return |
| \t | ASCII Horizontal Tab |
| \v | ASCII Vertical Tab |
| \ooo | Character with octal value ooo |
| \xHH | Character with hexadecimal value HH |

>>> print("C:\\Python32\\Lib")

C:\Python32\Lib

>>> print("This is printed\nin two lines")

This is printed

in two lines

>>> print("This is \x48\x45\x58 representation")

This is HEX representation

### Raw String to ignore escape sequence

Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place r or R in front of the string. This will imply that it is a raw string and any escape sequence inside it will be ignored.

>>> print("This is \x61 \ngood example")

This is a

good example

>>> print(r"This is \x61 \ngood example")

This is \x61 \ngood example

## Common Python String Methods

There are numerous methods available with the string object. The format() method that we mentioned above is one of them. Some of the commonly used methods are lower(), upper(), join(), split(), find(), replace() etc. Here is a complete list of all the [built-in methods to work with strings in Python](https://www.programiz.com/python-programming/methods/string).

>>> "SUchit".lower()

'suchit'

>>> " SUchit ".upper()

'SUCHIT'

>>> "This will split all words into a list".split()

['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']

>>> ' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])

'This will join all words into a string'

>>> 'Happy New Year'.find('ew')

7

>>> 'Happy New Year'.replace('Happy','Brilliant')

'Brilliant New Year'