Structured of Programming Language

Using Python

Introduction to Python Programming

Objectives

At the end of the module the student is expected to:

- **Define Python.**
- **List and explain the features of Python.**
- **List what Python can do.**
- **Compare Python syntax to other Programming Language**
- **Understand the History of Python**
- **List the different versions of Python**
- **Install Python**

Topics

- Define Python.
- Features of Python.
- What Python can do.
- Comparison Python syntax to other Programming Language
- History of Python
- Versions of Python
- Install Python

Introduction to Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas the other languages use punctuations. It has fewer syntactical constructions than other languages.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Python

- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python

- Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language: Python is a great language for the beginner level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python Features

Python's features include-

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows a student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintain.

Python Features

- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode: Python has support for an interactive mode, which allows interactive testing and debugging of snippets of code.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Python Features

- **Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.**

- **Databases: Python provides interfaces to all major commercial databases.**

Python Features

- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

- Scalable: Python provides a better structure and support for large programs than shell scripting.

Python Features

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

What can Python do?

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

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

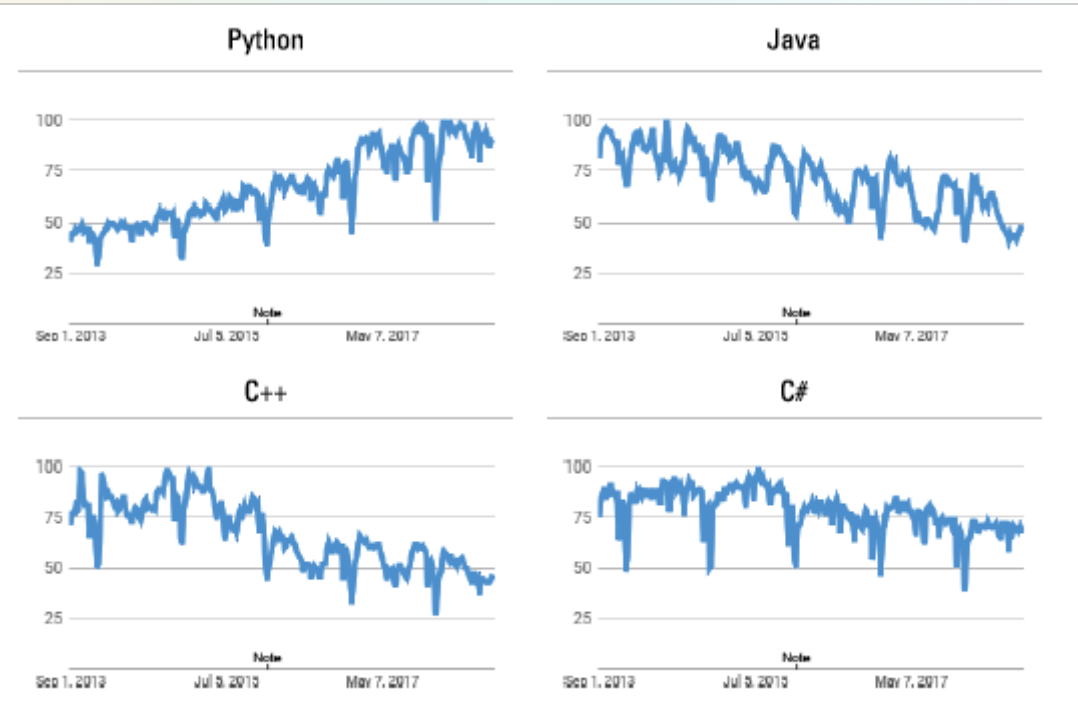
- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python Trends

Google search trends for the last five years or so.



History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

History of Python

- Python 1.0 was released in November 1994. In 2000, Python 2.0 was released. Python 2.7.11 is the latest edition of Python 2.
- Meanwhile, Python 3.0 was released in 2008. Python 3 is not backward compatible with Python 2. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules so that "There should be one – and preferably only one -- obvious way to do it." Python 3.8.3 is the latest version of Python 3.

Python Versions and Release Dates

Version	When Released
Python 3.7	June 2018
Python 3.6	December 2016.
Python 3.5	September 2015
Python 3.4	March 2014
Python 3.3	September 2012
Python 3.2	February 2011
Python 3.1	September 2012
Python 3.0	December 2008
Python 2.7	July 2010
Python 2.6	October 2008
Python 2.0	October 2000.
Python 1.6	September 2000.
Python 1.5	February 1998
Python 1.0	January 1994

Installing Python

Watch how to install python in

Windows Operating System

Mac Operating System

Syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the off-side rule, which some other languages share, but in most languages, indentation doesn't have any semantic meaning.

Statements and control flow

The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of *variables*) illuminates many other features of the language. Assignment in C, Example, **x = 2**, translates to "typed variable name *x* receives a copy of numeric value 2".

Statements and control flow

- The **if** statement, which conditionally executes a block of code, along with **else** and **elif** (a contraction of else-if).
- The **for** statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The **while** statement, which executes a block of code as long as its condition is true.
- The **try** statement, which allows exceptions raised in its attached code block to be caught and handled by **except** clauses; it also ensures that clean-up code in a **finally** block will always be run regardless of how the block exits.
- The **raise** statement, used to raise a specified exception or re-raise a caught exception.

Statements and control flow

- The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The def statement, which defines a function or method.
- The with statement, from Python 2.5 released in September 2006, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.
- The break statement, exits from the loop.
- The continue statement, skips this iteration and continues with the next item.

Statements and control flow

- The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The assert statement, used during debugging to check for conditions that ought to apply.
- The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.
- The import statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import * or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ...
- The print statement was changed to the print() function in Python 3.

Expressions

Some Python expressions are similar to languages such as C and Java, while some are not:

- Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating point/division. Python also added the ** operator for exponentiation.
- From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.
- From Python 3.8, the syntax :=, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.

Expressions

- In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c.
- Python uses the words and, or, not for its boolean operators rather than the symbolic &&, ||, ! used in Java and C.
- Python has a type of expression termed a *list comprehension*. Python 2.4 extended list comprehensions into a more general expression termed a *generator expression*.

Expressions

- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.
- Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).

Expressions

- Python makes a distinction between lists and tuples. Lists are written as **[1, 2, 3]**, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as **(1, 2, 3)**, are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The **+** operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable **t** initially equal to **(1, 2, 3)**, executing **t = t + (4, 5)** first evaluates **t + (4, 5)**, which yields **(1, 2, 3, 4, 5)**, which is then assigned back to **t**, thereby effectively "modifying the contents" of **t**, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.

Expressions

- Python features *sequence unpacking* wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an *iterable* object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.

Expressions

Python has a "string format" operator **%**. This functions analogous to **printf** format strings in C, e.g. **"spam=%s eggs=%d" % ("blah", 2)** evaluates to **"spam=blah eggs=2"**. In Python 3 and 2.6+, this was supplemented by the **format()** method of the **str** class, e.g. **"spam={0} eggs={1}".format("blah", 2)**. Python 3.6 added "f-strings": **blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'**.

Expressions

Python has various kinds of string literals:

- Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".
- Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
- Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.

Expressions

- Python has array index and array slicing expressions on lists, denoted as **a[key]**, **a[start:stop]** or **a[start:stop:step]**. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the *start* index up to, but not including, the *stop* index. The third slice parameter, called *step* or *stride*, allows elements to be skipped and reversed. Slice indexes may be omitted, for example **a[:]** returns a copy of the entire list. Each element of a slice is a shallow copy.

Expressions

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

- List comprehensions vs. for-loops
- Conditional expressions vs. if blocks
- The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Methods

Methods on objects are functions attached to the object's class; the syntax **instance.method(argument)** is, for normal methods and functions, syntactic sugar for **Class.method(instance, argument)**. Python methods have an explicit **self** parameter to access instance data, in contrast to the implicit **self** (or **this**) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Typing

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, **SpamClass()** or **EggsClass()**), and the classes are instances of the metaclass **type** (itself an instance of itself), allowing metaprogramming and reflection.

Typing

Before version 3.0, Python had two kinds of classes: *old-style* and *new-style*. The syntax of both styles is the same, the difference being whether the class **object** is inherited from, directly or indirectly (all new-style classes inherit from **object** and are instances of **type**). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

For Complete list visit
https://en.wikipedia.org/wiki/Python_(programming_language)#:~:text=Python%20was%20conceived%20in%20the%20late%201980s%20by%20Guido%20van,implementation%20began%20in%20December%201989
.

Python 3 Basic Sytax

Objectives

At the end of the module the student is expected to:

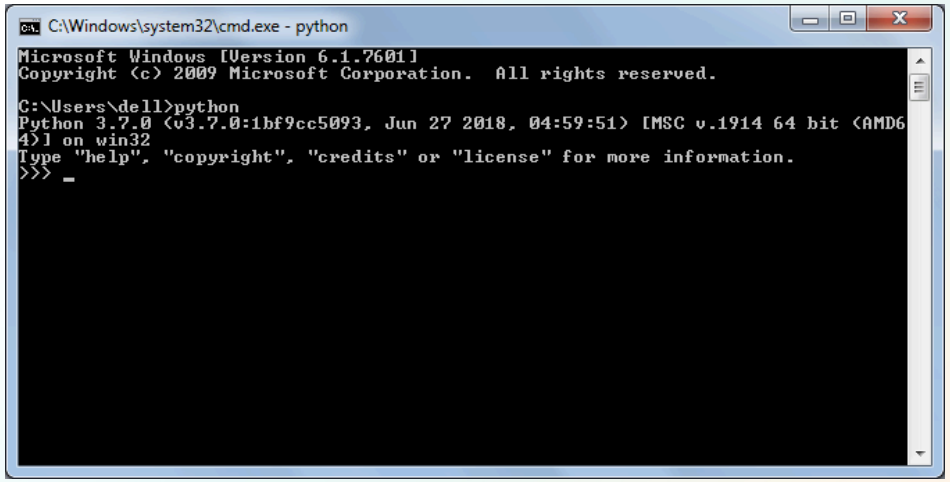
Execute Python commands using Shell
Execute Python script
Understand the Basic Syntax of Python
List and discuss the Keywords available in
Discuss Python Identifiers
Understand the use of Indents, and comments in Python
Understand and apply getting User input
Understand the use of Variables and Naming convention
Enumerate Python Data Types
Enumerate the different Comparison and Logical Operators in Python

Topics

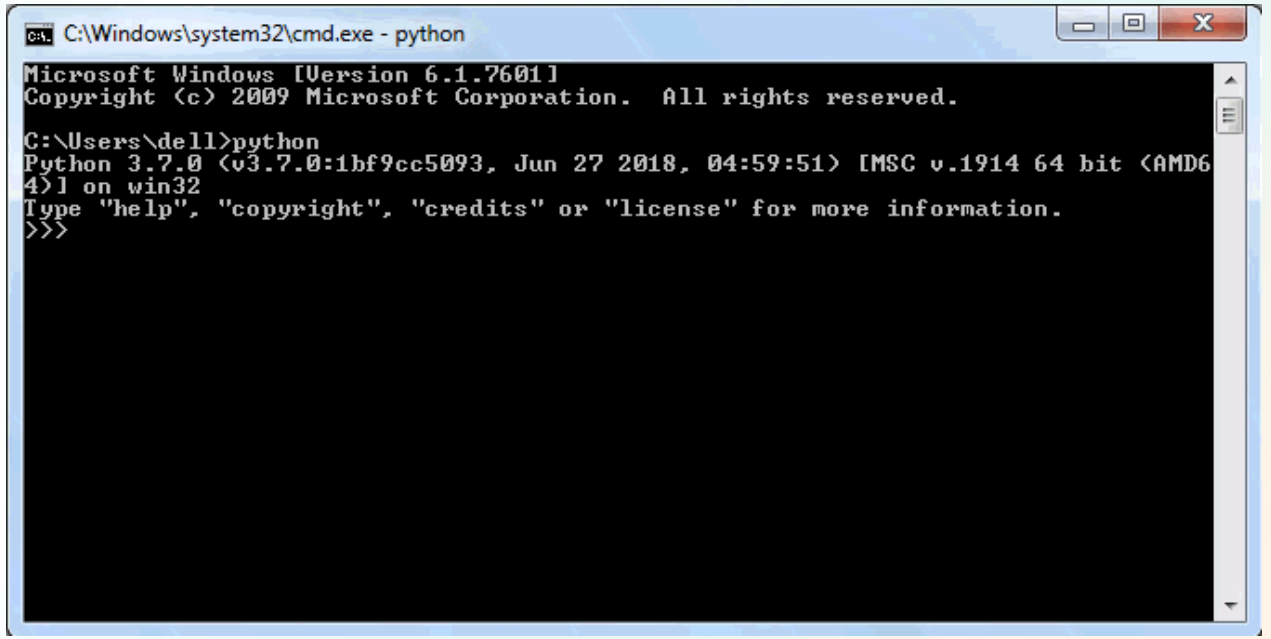
Python Shell - Interpreter
Executing Python Script
Basic Syntax
Keywords
Python Identifiers
Indents in Python
Comments in Python
Getting User input
Variables
Naming Conventions
Python Data Types
Comparison and Logical Operators in Python

Python - Shell (Interpreter)

- Python is an interpreter language. It means it executes the code line by line. Python provides a Python Shell (also known as Python Interactive Shell) which is used to execute a single Python command and get the result.
- Python Shell waits for the input command from the user. As soon as the user enters the command, it executes it and displays the result.
- To open the Python Shell on Windows, open the command prompt, write python and press **enter**.



As you can see, a Python Prompt comprising of three Greater Than symbols (>>>) appears. Now, you can enter a single statement and get the result. For example, enter a simple expression like **3 + 2**, press enter and it will display the result in the next line, as shown below.



Execute Python Script

As you have seen above, Python Shell executes a single statement. To execute multiple statements, create a Python file with extension .py, and write Python scripts (multiple statements).

For example, enter the following statement in a text editor such as Notepad.

Save the file as myPythonScript.py, to run python myPythonScript.py

```
print ("This is Python Script.")
print ("Welcome to Python Tutorial by Hadji Tejuco")
```

```
This is Python Script.
Welcome to Python Tutorial by Hadji Tejuco
```

Basic Syntax

Just like natural languages, a computer programming language comprises of a set of predefined words which are called keywords. A prescribed rule of usage for each keyword is called a syntax.

Python 3.x interpreter has 33 keywords defined in it. Since they have a predefined meaning attached, they cannot be used for any other purpose. The list of Python keywords can be obtained using the following help command in Python shell.

```
>>>help("keywords")
```

Keywords

The following table list all the keywords in Python.

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

Except for the first three (False, None and True), the other keywords are entirely in lowercase

Python Identifiers

Apart from keywords, a Python program can have variables, functions, classes, modules, packages etc. Identifier is the name given to these programming elements. An identifier should start with either an alphabet letter (lower or upper case) or an underscore (_). After that, more than one alphabet letters (a-z or A-Z), digits (0-9) or underscores may be used to form an identifier. No other characters are allowed.

Python Identifiers

- Conventionally, the name of the class begins with an uppercase alphabet letter. Others start with lowercase alphabet letters.
- Use of one or two underscore characters has a special significance when naming the instance attributes of a class. More about this will follow in the discussion about inheritance.
- Two leading and trailing underscores are used in the language itself for a special purpose. For example __add__, __init__

Note: identifiers are case sensitive which means variables named age and Age are different. Identifiers can be of any length.

Python Statement

By default, the Python interpreter treats a piece of text terminated by hard carriage return (new line character) as one statement. It means each line in a Python script is a statement. (Just as in C/C++/C#, a semicolon ; denotes the end of a statement).

```
msg="Hello World"
code=123
name="Hadji"
```

Python Statement

However, you can show the text spread over more than one lines to be a single statement by using the backslash (\) as a continuation character. Look at the following examples:

Example: Continuation of Statement

```
msg="Hello Pythonista \
Welcome to Python Tutorial Series \
from Hadji Tejuco"
```

Python Statement

Similarly, use the semicolon ; to write multiple statements in a single line.

Example: Multiple Statements in Single Line

msg="Hello World";code=123;name="Hadji"

Indents in Python

Many times it is required to construct a block of more than one statements. For example there are usually multiple statements that are part of the definition of a function. There can be one or more statements in a looping construct.

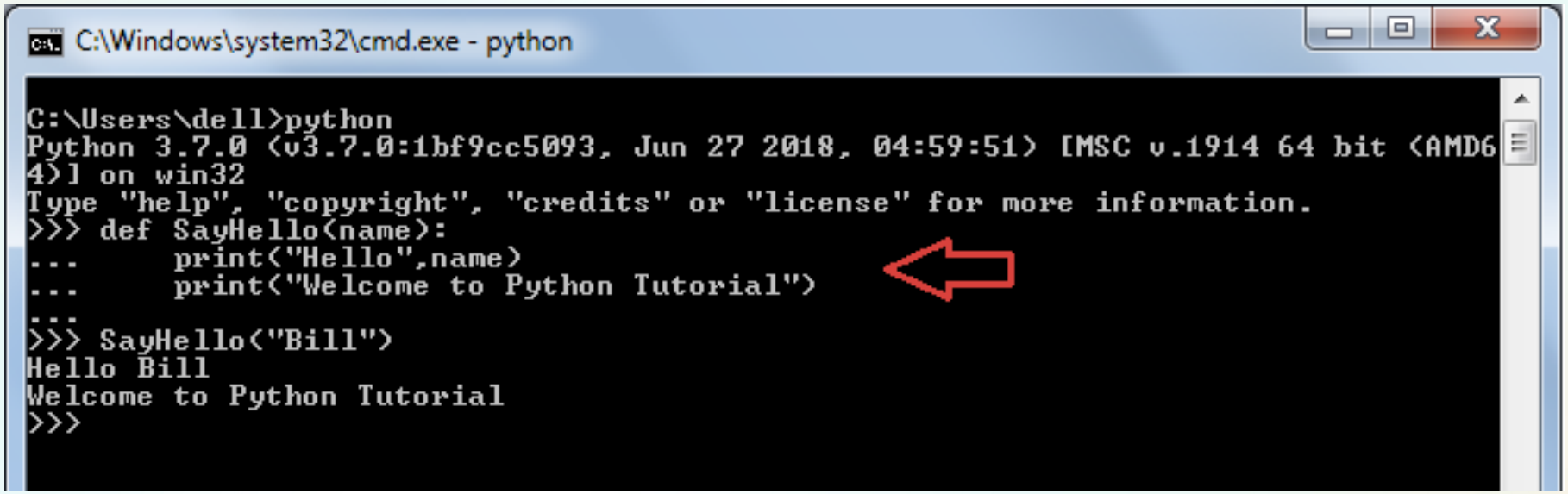
Different programming languages use different techniques to define the scope and extent of a block of statements in constructs like class, function, conditional and loop. In C, C++, C# or Java, statements inside curly brackets { and } are treated as a block.

Indents in Python

Python uses uniform indentation to denote a block of statements. When a block is to be started, type the exclamation symbol (:) and press Enter. Any Python-aware editor (like IDLE) goes to the next line leaving an additional whitespace (called indent). Subsequent statements in the block follow the same level of indent. In order to signal the end of a block, the whitespace is de-dented by pressing the backspace key. If your editor is not configured for Python, you may have to ensure that the statements in a block have the same indentation level by pressing the **spacebar** or **Tab** key. The Python interpreter will throw an error if the indentation level in the block is not same.

Indents in Python

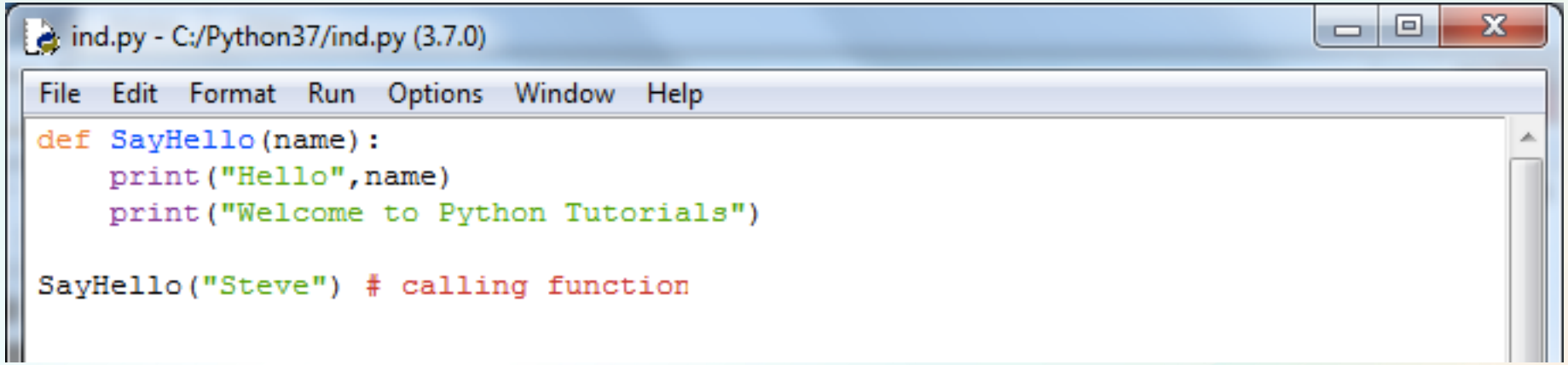
The following example illustrates the use of indents in Python shell:



Indents in Python

As you can see, in the Python shell, the SayHello() function block started after : and pressing Enter. It then displayed ... to mark the block. Now, use **Tab** for indent and then write a statement. To end the block, press Enter two times.

The same function can be written in IDLE or any other GUI-based IDE as shown below, using **Tab** as indentation.



Comments in Python

In a Python script, the symbol # indicates the start of a comment line. It is effective till the end of the line in the editor. If # is the first character of the line, then the entire line is a comment. It can be used also in the middle of a line. The text before it is a valid Python expression, while the text following is treated as a comment.

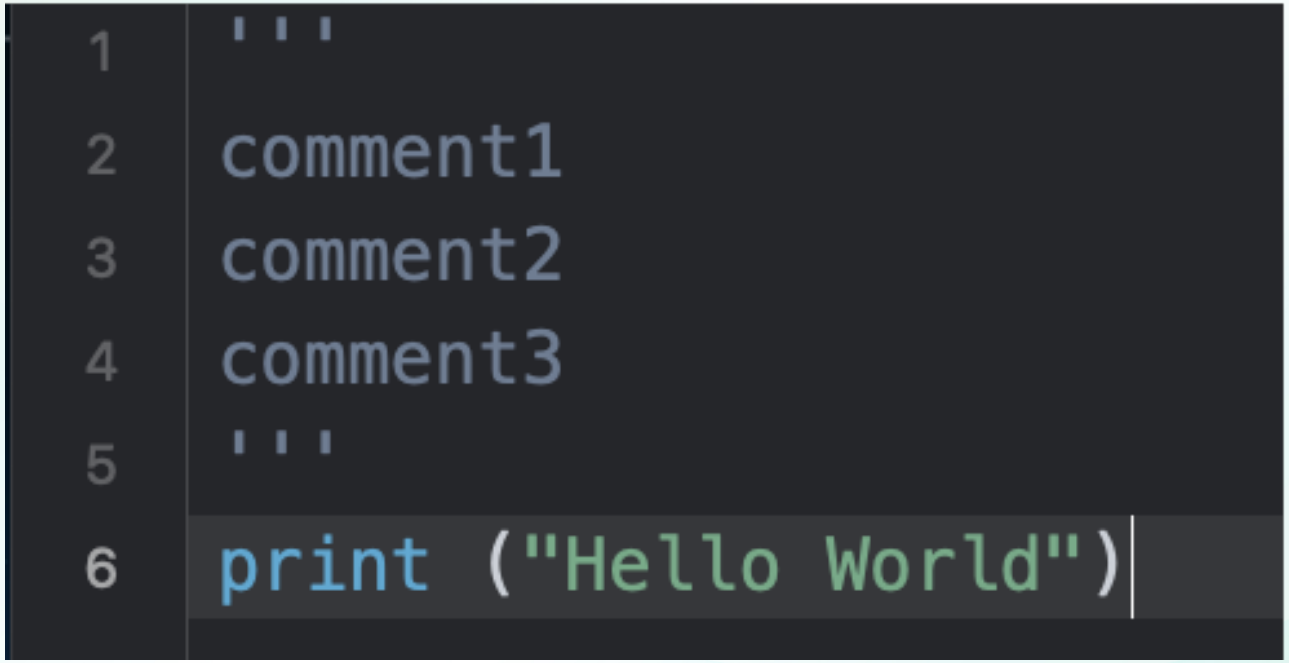
```
# this is a comment
print ("Hello World")
print ("Welcome to Python Tutorial") #this is also a comment but after a statement.
```

Comments in Python

In Python, there is no provision to write multi-line comments, or a block comment. (As in C#/C/C++, where multiple lines inside /* .. */ are treated as a multi-line comment). Each line should have the # symbol at the start to be marked as a comment. Many Python IDEs have shortcuts to mark a block of statements as a comment. In IDLE, select the block and press Alt + 3.

Comments in Python

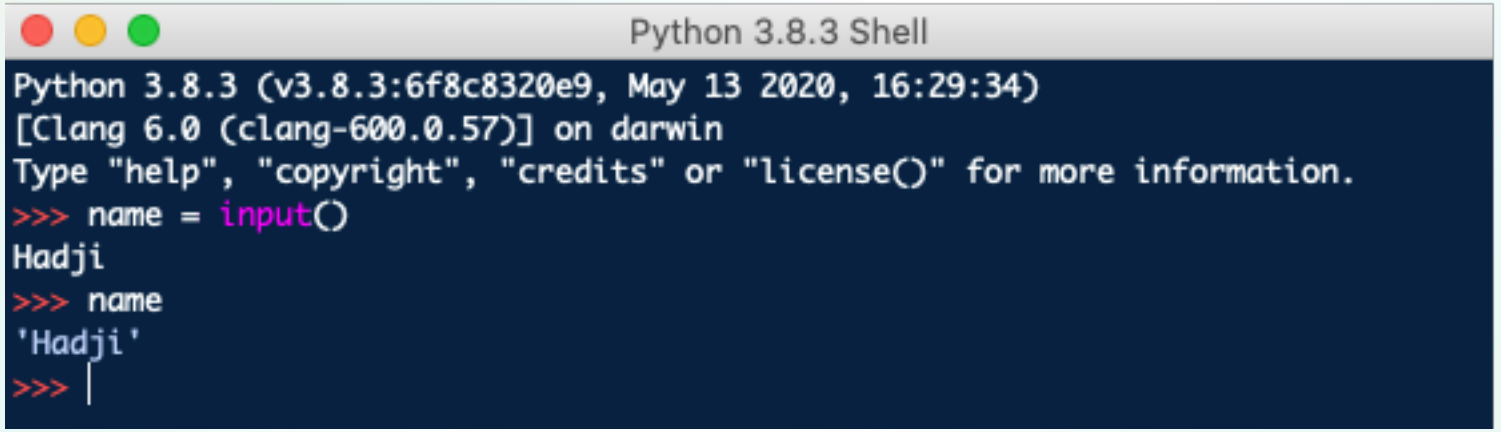
A triple quoted multi-line string is also treated as a comment if it is not a docstring of a function or a class. (The use of docstring will be explained in subsequent tutorials on Python functions.)



Getting User Input

Getting the User's Input

The **input()** function is a part of the core library of standard Python distribution. It reads the key strokes as a string object which can be referred to by a variable having a suitable name.

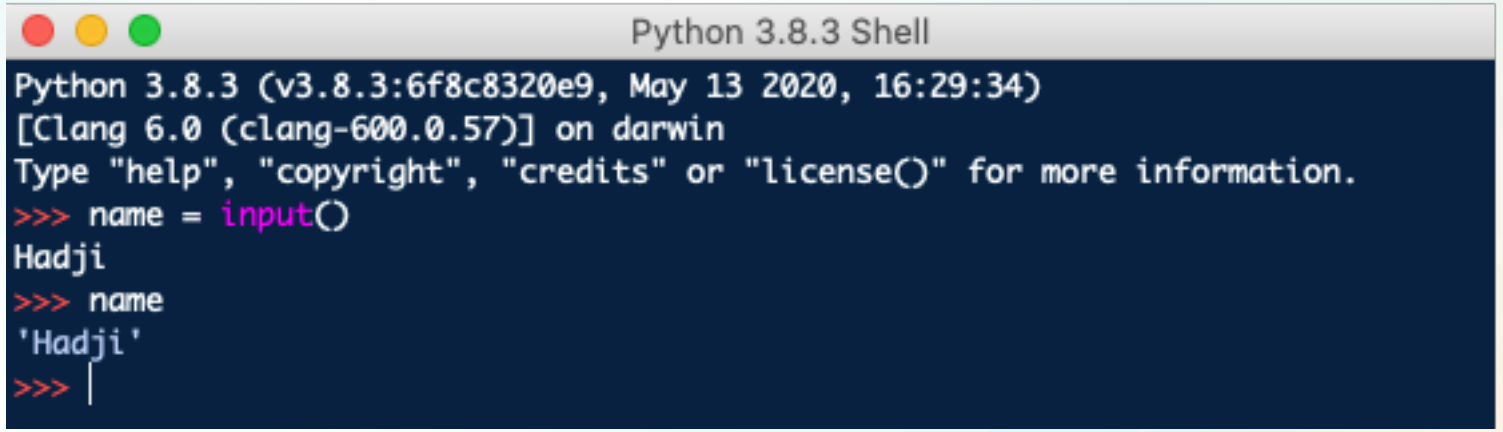


Note that the blinking cursor waits for the user's input. The user enters his input and then hits Enter. This will be captured as a string.

Getting the User's Input

In the above example, the input() function takes the user's input from the next line, e.g. ‘Hadji’ in this case. input() will capture it and assign it to a name variable. The name variable will display whatever the user has provided as the input.

The input() function has an optional string parameter that acts as a prompt for the user.



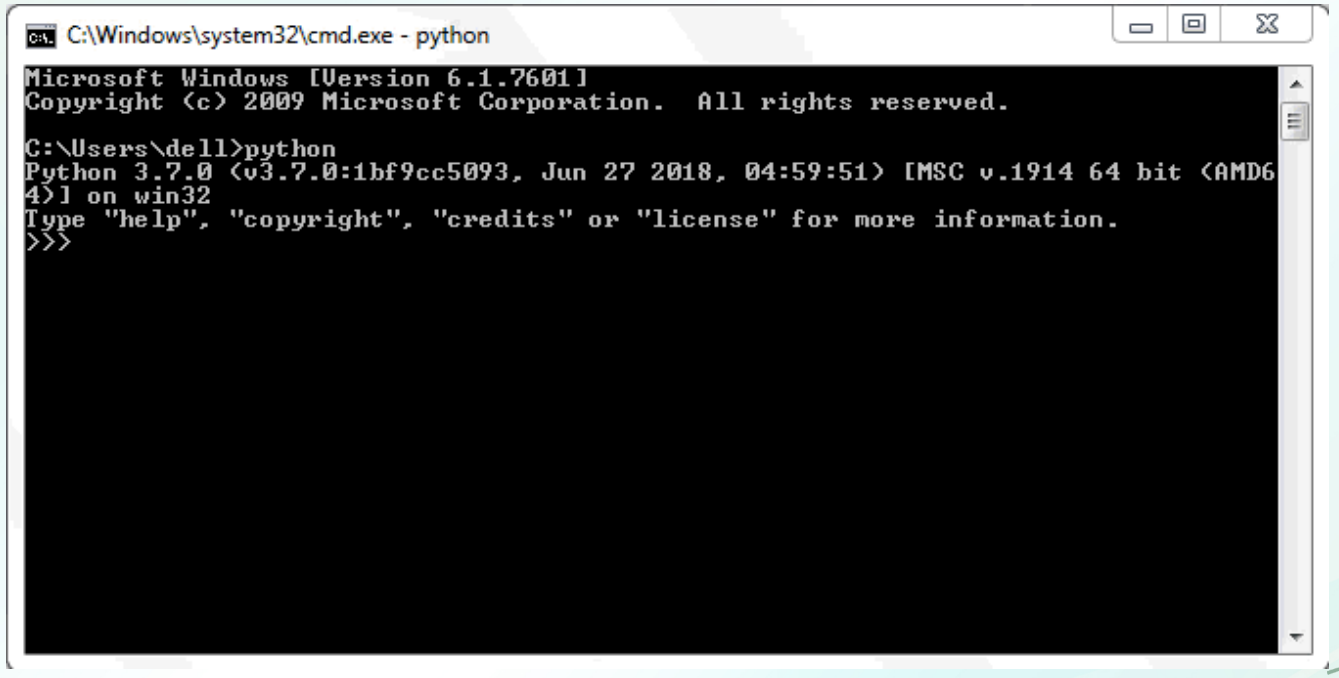
Getting the User's Input

The input() function always reads the input as a string, even if comprises of digits. The type() function used earlier confirms this behaviour.

```
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> name=input("Enter your name: ")
Enter your name: Hadji
>>> type(name)
<class 'str'>
>>> age = input ("Enter your age: ")
Enter your age: 15
>>> type(age)
<class 'str'>
>>>
```

Display the Output

Another built-in function print() serves as an output statement in Python. It echoes the value of any Python expression on the Python shell.



Multiple values can be displayed by the single print() function separated by comma. The following example displays values of name and age variables using the single print() function.

```
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> name="Hadji"
>>> age = 21
>>> print ("Name: ",name, "Age: ",age)
Name:  Hadji Age:  21
>>>
```

By default, a single space (' ') acts as a separator between values. However, any other character can be used by providing a **sep** parameter. In the following example, "=" is used as a separator character.

```
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> name="Hadji"
>>> age = 21
>>> print ("Name: ",name, "Age: ",age)
Name:  Hadji Age:  21
>>> print (name,age)
Hadji 21
>>> print (name,age,sep=",")
Hadji,21
>>>
```

The output of the print() function always ends by a newline character. The print() function has another optional parameter **end**, whose default value is "\n". This value can be substituted by any other character such as a single space (' ') to display the output of the subsequent print() statement in the same line. This is especially useful in a Python script like the one shown below:

Example

Example: display.py

```
1  name="Hadji"
2  age=21
3  print("Name:", name, end=" ")
4  print("Age:", age)

Name: Hadji Age: 21
```

Save the above script as display.py and open the command prompt in Windows or the terminal in another platform, and run the above script as below.

It is possible to format the output using C style format specifier symbols such as **%d, %f, %s** etc.
This will be discussed in the chapter on Strings.

Variables

Variables

- Any value of certain type is stored in the computer's memory for processing. Out of available memory locations, one is randomly allocated for storage. In order to conveniently and repeatedly refer to the stored value, it is given a suitable name. A value is bound to a name by the assignment operator ‘=‘. Example myVar=21
- Here myVar is an identifier (name) referring to integer value 21 (Python treats data value as a value). However, the same identifier can be used to refer to another value. For example, the below code will assign myVar as the name of a string value, as shown below. **>>> myVar=”Hadji"**

So, the value being referred can change (or vary), hence it is called a variable. It is important to remember that a variable is a name given to a value, and not to a memory location storing the value.

Variables can be accessed by using their identifier (name) as shown below.

```
>>> myVar=”Hadji Tejuco"
>>> myVar
‘Hadji Tejuco'
```

Dynamic Typing

One of the important features of Python is that it is a dynamically-typed language. Programming languages such as C, C++, Java, and C# are **statically-typed languages**. A variable in these languages is a user-friendly name given to a memory location and is intended to store the value of a particular data type.

Dynamic Typing

A variable in Python is not bound permanently to a specific data type. Instead, it only serves as a label to a value of a certain type. Hence, the prior declaration of variable's data type is not possible. In Python, the data assigned to a variable decides its data type and not the other way around.

Dynamic Typing

In the following Python statements, a string value is assigned to a variable 'name'. We can test its type using the **type()** function.

```
>>> name=”Hadji Tejuco"
>>> type(name)
<class 'str'>
```

Dynamic Typing

Now the Python interpreter won't object if the same variable is used to store a reference to an integer.

```
>>> name=1234
>>> type(name)
<class 'int’>
```

We can see that the data type of a variable has now been changed to integer. This is why Python is called a **dynamically-typed language**.

Naming Conventions

Any suitable identifier can be used as a name of a variable, based on the following rules:

- The name of the variable should start with either an alphabet letter (lower or upper case) or an underscore (_), but it cannot start with a digit.
- More than one alpha-numeric characters or underscores may follow.
- The variable name can consist of alphabet letter(s), number(s) and underscore(s) only. For example, myVar, MyVar, _myVar, MyVar123 are valid variable names but m*var, my-var, 1myVar are invalid variable names.
- Identifiers in Python are case sensitive. So, NAME, name, nAME, and nAmE are treated as different variable names.

Starting the name with a single or double underscore has a special meaning in Python.

Python Data Types

Data Types

Data types are the classification or categorization of data items. Data types represent a kind of value which determines what operations can be performed on that data. Numeric, non-numeric and Boolean (true/false) data are the most used data types. However, each programming language has its own classification largely reflecting its programming philosophy.

Numeric

Python has the following standard or built-in data types:

A numeric value is any representation of data which has a numeric value. Python identifies three types of numbers:

- **Integer:** Positive or negative whole numbers (without a fractional part)
- **Float:** Any real number with a floating point representation in which a fractional component is denoted by a decimal symbol or scientific notation
- **Complex number:** A number with a real and imaginary component represented as x+yj. x and y are floats and j is -1(square root of -1 called an imaginary number)

Boolean

Data with one of two built-in values **True** or **False**. Notice that 'T' and 'F' are capital. **true** and **false** are not valid booleans and Python will throw an error for them.

Sequence Type

A sequence is an ordered collection of similar or different data types. Python has the following built-in sequence data types:

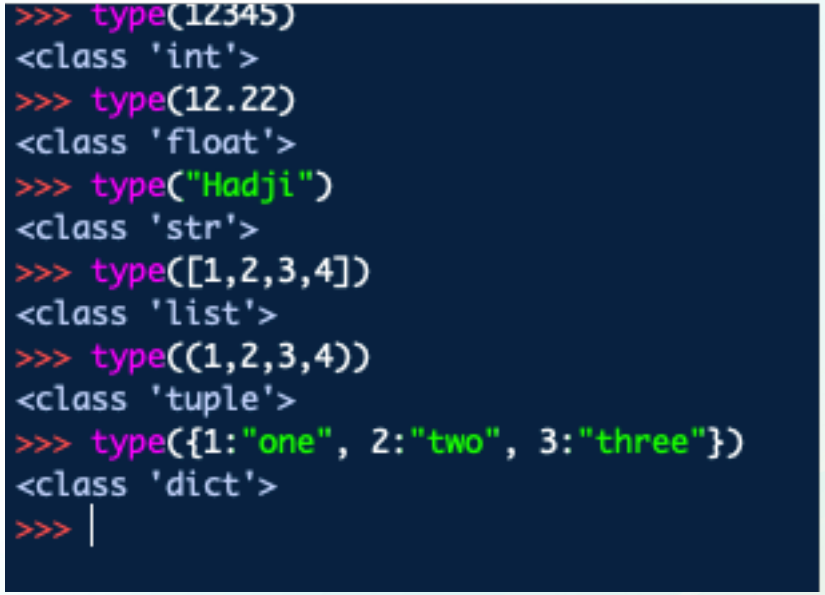
- **String**: A string value is a collection of one or more characters put in single, double or triple quotes.
- **List** : A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.
- **Tuple**: A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses.

Dictionary

A dictionary object is an unordered collection of data in a key:value pair form. A collection of such pairs is enclosed in curly brackets. For example: **{1:”Hadji", 2:”Joan", 3:”Jeieven", 4: ”Johann"}**

type() function

Python has an in-built function **type()** to ascertain the data type of a certain value. For example, enter **type(1234)** in Python shell and it will return **<class 'int'>**, which means 1234 is an integer value. Try and verify the data type of different values in Python shell, as shown below.



Mutable and Immutable Objects

Data objects of the above types are stored in a computer's memory for processing. Some of these values can be modified during processing, but contents of others can't be altered once they are created in the memory.

Number values, strings, and tuple are immutable, which means their contents can't be altered after creation.

On the other hand, collection of items in a List or Dictionary object can be modified. It is possible to add, delete, insert, and rearrange items in a list or dictionary. Hence, they are mutable objects

Operators in Python

Python Arithmetic Operators

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

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

Python Assignment Operators

Assignment operators are used to assign values to variables:

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

Comparison and Logical Operators in Python

The comparison operators returns a boolean either True or False.

Assuming that x=10 and y=20, the result of the operations is also given in following table:

Operator	Description	Example
>	True if the left operand is higher than the right one	>>> x>y False
<	True if the left operand is lower than right one	>>> x<y True
==	True if the operands are equal	>>> x==y False
!=	True if the operands are not equal	>>> x!=y True
>=	True if the left operand is higher than or equal to the right one	>>> x>=y False
<=	True if the left operand is lower than or equal to the right one	>>> x<=y True

Logical Operators in Python

The following keywords in Python combine two Boolean expressions. They are called logical operators. Two operands should have Boolean value True or False. Assuming that x=True and y=False.

Operator	Description	Example
and	True if both are true	>>> x and y False
or	True if at least one is true	>>> x or y True
not	Returns True if an expression evalutes to false and vice-versa	>>> not x False

Comparison and logical operators are useful in controlling flow of program.

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
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

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