[**Types of Python Environments**](#_4t3qn8t1afvu)[**3**](#_4t3qn8t1afvu)

[Python Shell Interpreter (Pre-Installed): Command Line or Terminal](#_fsjt25lpu8gj) [3](#_fsjt25lpu8gj)

[Installed Environments (Navigator / Anaconda: Spyder)](#_ojt6f6196plr) [3](#_ojt6f6196plr)

[Installed Browser Environment (Anaconda: Jupyter /](#_jgkeyjs5gbhl) [4](#_jgkeyjs5gbhl)

[Text Editors (Atom / Gedit / Emacs/ Notepad++ / Brackets)](#_lyafanlq3eqg) [4](#_lyafanlq3eqg)

[**Program Syntax**](#_133wsf3ey881)[**4**](#_133wsf3ey881)

[Identifying keys in dictionaries](#_6cmey8swtuw1) [5](#_6cmey8swtuw1)

[Writing Functions](#_1dj47feakrwf) [5](#_1dj47feakrwf)

[Delimiter](#_2rgaysnr71sm) [5](#_2rgaysnr71sm)

[**Variables**](#_77wkx9v7d5pu)[**6**](#_77wkx9v7d5pu)

[**Data Types**](#_yvgh41elvyvl)[**8**](#_yvgh41elvyvl)

[Integers](#_doejt0c1qned) [8](#_doejt0c1qned)

[Strings](#_qeupa1zgzmdk) [8](#_qeupa1zgzmdk)

[Lists](#_ixme9vpzow9d) [8](#_ixme9vpzow9d)

[Tuples](#_e47ox19brqf6) [9](#_e47ox19brqf6)

[Dictionary](#_jgo2fj6jae39) [9](#_jgo2fj6jae39)

[**Program Flow**](#_l6in9nmnru0e)[**9**](#_l6in9nmnru0e)

[**Operators**](#_gjdgxs)[**10**](#_gjdgxs)

[Arithmetic](#_9i50nab88m0y) [10](#_9i50nab88m0y)

[Comparison](#_wr4ad819oox2) [10](#_wr4ad819oox2)

[Assignment](#_t448c37hekwy) [10](#_t448c37hekwy)

[Membership](#_egjzaqn6u5kn) [11](#_egjzaqn6u5kn)

[Logical](#_okm4secejjuc) [11](#_okm4secejjuc)

[**Control Flow**](#_7xbeius3e2ci)[**12**](#_7xbeius3e2ci)

[**Loops**](#_j1cql1doozlj)[**12**](#_j1cql1doozlj)

[**Arrays**](#_8srhjpb9amii)[**13**](#_8srhjpb9amii)

[**Importing Packages**](#_jab0v4uv29mj)[**13**](#_jab0v4uv29mj)

[**Functions**](#_bd5v7ogy20uq)[**15**](#_bd5v7ogy20uq)

[**Object Oriented**](#_5cp997mkvxgp)[**17**](#_5cp997mkvxgp)

[**Extra**](#_3878wsb13jad)[**19**](#_3878wsb13jad)

## Types of Python Environments

### Python Shell Interpreter (Pre-Installed): Command Line or Terminal

The Python interpreter is usually installed as /usr/local/bin/python on those machines where it is available; putting /usr/local/bin in your Unix shell’s search path makes it possible to start it by typing the command

MAC OS: <https://www.python.org/downloads/mac-osx/>

WINDOWS: <https://www.python.org/downloads/windows/>

### Installed Environments (Navigator / Anaconda: Spyder)

A more traditional development environment. Larger projects generally use something more for interactive analytics and prototyping. Rapid interactive development with full fledged editor, more powerful project management and debugging capabilities.

Python 2 vs 3

https://docs.python.org/3/whatsnew/3.0.html

### Installed Browser Environment (Anaconda: Jupyter )

Javascript based web browser version that is widely used by professionals for interactive analytics and for teaching and presenting. Contains Magic commands. No editor built directly into the application.

### Text Editors (Atom / Gedit / Emacs/ Notepad++ / Brackets)

Program used to edit plain text files for software development.

## Program Syntax

Quotation Marks

Single (‘), Double (“) and triple (‘’’ or “””) quotes to denote strings are all acceptable as long as the same type of quote begins and ends the string.

Note: Triple Quotes are used for strings beyond 1 line.

Indents

Python code does not have any braces to indicate blocks of code. Blocks of code are denoted by line indentation. the number of spaces can vary but all statements must be indented the same amount.

e.g. if True:

print “True”

else:

print “False”

e.g. if True:

print “Answer”

print “True”

else:

print “Answer”

print “False”

Brackets

[]: used for mutable data types – lists, and for lookup/indexing/slicing

(): Define tuples, operation orders, generator expressions, and function calls

{}: used for hash tables – dictionaries and sets

Colons

Colons can be used in 3 ways.

### Identifying keys in dictionaries

Maps a key to a value.

### Writing Functions

Used to declare start of an indented block in a function.

### Delimiter

Slicing out sub-parts in sequences [start:end]

e.g. [1:5] is “from 1 to 4”

[1:] is “1 to end”

[len(a):] is “from length of a to end”

Semi-Colons

Used to create multiple statements on a single line.

e.g. import sys; x = foo; sys.stdout.write(x + ‘\n’)

Comments

A good practice is to write comment in your code.

Comments can be written using

“#” – one lined comment

## Variables

Variables are reserved memory locations to store values. When you create a variable, you reserve some space in memory. Different data types have different memory allocation. You assign values to variables using “=”

Assignment: The name on the left-hand side refers to the result of evaluating the right hand side.

e.g. counter = 100 # assigns an integer

miles = 1000.0 # assigns a float

name = “John” # assigns a string

Multi-Assignment:

single value to several variables,

A = b = c = 1

or multiple objects to multiple variables

A, b, c = 1, 2, “john”

Delete variables

Delete reference to an object by using del

e.g. del var1[,var2[,var3[…,varN]]]]

Delete single or multiple objects

e.g. del var

del var\_a, var\_b

Class of Objects

Class Example:

Data = 42

Def method(self):pass

Creates an object named Example, representing the class of all things that are Examples.

Scope

All variables may not be accessible at all locations. This depends on where the variable is declared. There are 2 types: Global and local variables. Local variables can only be accessed inside the function while global variables can be accessed by any function in the program.

Global Variables:

Variables that are defined outside of a function body.

Local Variables:

Variables that are defined inside the function.

To define a variable in the local namespace from the global namespace, use the global statement to make use of the global assignment.

e.g. Money = 2000

def AddMoney():

global Money

Money = Money + 1

## Data Types

Five data types exist in Python. To know what kind of data type you are using, use type(var)

### Integers

Stores numeric values. Four different types are int (signed integers), long (long integers), float (floating point real numbers), complex (complex numbers). Arithmetic operators are affected by the highest order int type.

e.g. a = 10

b = 51924361 / 0xDEFABCECBDAECBFBAEI

c = 15.20 / 32.3+e18

d = 3e + 26J (an ordered pair of real floating-point numbers by x + yj)

### Strings

Continuous set of characters represented in quotation marks. Index of first element begins at 0 from the start and -1 from the end. Strings are commonly referred to as Characters with length one.

e.g. str = “Hello World!”

Note: “123” =/= 123

### Lists

Items separated by commas and enclosed by square brackets []. Similar to arrays but items in a list can be of different data types. Index of first element beings at 0 from the start and -1 at the end. Lists are mutable.

e.g. list = [‘abcd’, 123, 4.47, “john”]

### Tuples

Similar to list. However, tuples are enclosed in parentheses () and are immutable (cannot be updated or changed using different operations.)

e.g. tuple = (‘abcd’, 786, 4.47, ‘john’)

### Dictionary

A kind of associative array or hash like hash table. It consists of key-value pairs enclosed by {}. Any object can be a key. Used for data retrieval by keys. Dicts are unordered and unsortable but mutable.

e.g. dict = { ‘Name’ : ‘Angela merkel’,

‘Country’ : ‘Germany’,

‘Profession’ : ‘Chancelor’,

‘Age’ : 60 }

## Program Flow

## Operators

Types: Arithmetic, Comparison, Assignment, Logical, Bitwise, Membership, Identity

### Arithmetic

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| $ | Modulus (Returns the remainder) |
| \*\* | Exponent |
| // | Floor Division (Digits after the decimal point are removed) |

### Comparison

|  |  |
| --- | --- |
| == | If value of left and right are equal, condition is true. |
| != | If value of left and right are not equal, condition is true |
| > | If value of left is greater than right, condition is true |
| < | If value of left is less than right, condition is true |
| >= | If value of left is greater or equal than right, condition is true |
| <= | If value of left is less or equal than right, condition is true |

### Assignment

|  |  |
| --- | --- |
| += | Add and assign the right result to the left |
| -= | Subtract and assign the right result to the left |
| \*= | Multiply and assign the right result to the left |
| /= | Divide and assign the right result to the left |
| %/ | Modulus and assign the right result to the left |
| \*\*/ | Exponent and assign the right result to the left |
| //= | Floor divide and assign the right result to the left |

### Membership

|  |  |
| --- | --- |
| In | True if a variable is a member of the specified sequence |
| not in | True if a variable is not a member of the specified sequence |

### Logical

|  |  |
| --- | --- |
| And (&) | If both operands are true then condition is true |
| Or (|) | If either operand is true then condition is true |
| Not | Reverse the logical state of the operand |

## Control Flow

Control flow is a result of decision making by evaluating multiple expressions to produce TRUE or FALSE as outcomes.

If..Else

An if statement executes when the expression given is true, and the else executes when the expression is false.

e.g. if 2>3:

print(“True”)

else:

print(“False”)

## Loops

While

Repeats a statement while a given condition is TRUE. Condition is tested before executing the loop.

def countdown(n):

while n > 0:

print n

n = n-1

print ' Blastoff!'

For

Executes a sequence of statements multiple times.

Loop Control Statements: Change execution from normal sequence. (Use at your own risk)

|  |  |
| --- | --- |
| Break | Terminates loop statement and goes to the preceding statement |
| Continue | Skip the remainder of the loop and immediately retest its condition prior to reiterating |
| Pass | Used when statement is required syntactically but hasn’t been fully written or you do not want the command to execute |

## Arrays

Arrays don’t actually exist in Python. They are imported via NumPy under the ndarray class for Scientific and Engineering Applications.

ndarray was built with the goal of handling n-dimensional arrays. It is more powerful than lists and can be used to handle very large data sets efficiently, storing vectors, matrices and tensors. (Own Exploration)

## Importing Packages

Packages or modules are files consisting of Python code which define functions, classes, and variables. You can use any Python source file as a module by executing an import statement in some other Python source file.

The interpreter searches for the module in the search path and imports the module if it is present in the search path. It is only loaded once regardless of how many times you import it.

The search path for modules is in a) Current Directory, b) Each directory in PYTHONPATH, and c) The Default Path usually containing the entire Python folder.

Import Syntax

**Import module1[, module2[,… moduleN]**

e.g. import matplotlib

import numpy

import random

Saving Space and Time

If you know specifically which attributes from a module you would like to use, use the from…import statement. Rather than importing the entire module into your workspace, it introduces the specific item into the workspace.

**from modname import name1[, name2[,… nameN]]**

e.g. from fib import fibonacci

To see what attribute names are inside the module, use the dir() function.

e.g. import math

dir(math)

## Functions

Functions are blocks of organized, reusable code. They allow better modularity for application and high degree of code reusing.

Defining a Function

1. Begin with the keyword def functionname ( )
2. Input parameters or arguments within ( )
3. Documentation String (Optional)
4. Begin the code block with a colon (:)
5. Indent the code block
6. Write the code block
7. End function with a return expression. Return statement can have no arguments and returns None.

**def functionaname(parameters):**

**“function\_docstring”**

**#function goes here**

**return [expression]**

e.g. def printme(str)

“This prints a passed string into this function”

print str

return

Calling a Function

Call a function from another function, or from the prompt

e.g. printme(“Call a function”)

Anonymous Functions

They are not declared using def but using lambda.

1. They can take any number of arguments but return only one value in the form of an expression.
2. They cannot contain commands or multiple expressions.
3. They cannot be a direct print call.
4. Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and global namespace.

**lambda [arg1 [,arg2, … argn]]:expression**

e.g. map(lambda x: x \*\* 2, range(10))

=> [0, 1, 4, 9,16, 25,36,49,64, 81]

Recursive Functions

A function can call another function in Python. The idea is known as Recursion. Recursion allows a function to call itself or another function.

e.g. def countdown(n):

if n <= 0:

print ' Blastoff!'

else:

print n

countdown(n-1)

## Object Oriented

Python is an object-oriented language. It helps you to easily create classes and objects. Object-oriented programming is a module organized around objects and data rather than actions and logic. Computer programs are designed to interact with one another rather.

In OOP, we identify all objects to be manipulated and find out how they relate to one another. Once we have identified the object, we generalize it as a class of objects (e.g. all chairs are classified with the same class of properties of an ideal chair.)

In Python, these objects are known as classes. By creating a class, you are creating a parent branch with x attributes. You can easily assign future data into this parent branch which will inherit all attributes encoded inside the parent branch.

Syntax

class ClassName:

‘Documentation String’

#insert methods and code

e.g. class Employee:

‘Common base class for all employees’

empCount = 0

def \_init\_(self, name, salary):

self.name = name

self.salary = salary

Employee.empcount += 1

def displayCount(self):

print “Total employee %d” % employee.empCount

def displayEmployee(self):

print “Name : “, self.name, “, Salary: “, self.salary

Notes: \_init\_() must always be there when you create a new instance of a class.

Class Inheritance

Instead of starting from scratch, you can create a class using preexisting classes by listing the parent class in parentheses. The child inherits the attribute of the parent class.

Syntax

Class SubClassName (ParentClass1[, ParentClass2, …]);

‘Documentation String’

#insert methods and code

## Extra

**Data Type Conversion**

|  |  |
| --- | --- |
| Int(x [,base]) | Convert x to an integer. Base specifies the base if x is a string |
| long(x [,base]) | Convert x to a long integer. Base specifies the base if x is a string |
| float(x) | Convert x to a floating point number |
| str(x) | Convert x to a string representation |
| repr(x) | Convert x to an expression string |
| eval(x) | Evaluate a string and return an object |
| tuple(x) | Convert x to a tuple |
| list(x) | Convert x to a list |
| dict(x) | Create a dictionary where x must be a sequence of (key,value) tuples |
| frozenset(x) | Convert x to a frozen set |
| chr(x) | Convert integer to character |
| unichr(x) | Convert integer to Unicode character |
| ord(x) | Convert a single character to an integer |

**Data Types**

Sets

A set object is an unordered collection of mutable values. It has not hash value and cannot be used as a dictionary key.

Frozen Sets

The Frozenset is immutable and hashable. The contents cannot be altered after creation and can be used as a dictionary key of another set.

Unicode Character

Python’s default encoding is ASCII. With characters of ASCII value > 127, you will get UnicodeDecodeError because those characters can’t be handled by ASCII encoding[[1]](#footnote-0).

Characters represented by code points. Code points are integer values with base 16 from 0 to 0x10fff. Unicode Strings are used to represent different letters and symbols from other languages into one file.

Sequence of characters are represented as a set of bytpes through encoding. There are 2 types of Unicodes: 8-UTF (8-bit characters: 2^8 values, or 16-bit characters: 2^16 values.)

Hexadecimal Strings

A base 16 positional numeral system. Uses 16 distinct symbols from 0-9 and A-F to represent the values 0 to 16. Hexadecimal numbers are used to represent four binary bits and are represented by the prefix “0x”.

e.g. 0101 1110 1011 0101 0010 (base 2) OR 387922 (base 10) is the same as 5EB52 (base 16)

Octal Strings

A base 8 positional numeral system using 8 digits from 0 to 7.

e.g. 387922 (base 10) is the same as 1365522 (base 8).

**Reserved Words**

You cannot use these words as constants or variables or any other identifier names. The reserved words are only in lowercase.

|  |  |  |
| --- | --- | --- |
| and | exec | not |
| assert | finally | or |
| break | for | pass |
| class | from | print |
| continue | global | raise |
| def | if | return |
| del | import | try |
| elif | in | while |
| else | is | with |
| except | lambda | yield |

**Operator precedence**

|  |
| --- |
| \*\* |
| \* / % // |
| + - |
| & |
| | |
| <= < > >= |
| < > == != |
| = %= /= //= -= += \*= \*\*= |
| Is, is not |
| In, not in |
| not, or, and |

**Quick check**

Rather than place hundreds of if statements in your program, you can use an assertion to check the program input a user gives.

Syntax

Assert Expression[, Arguments]

e.g. def KelvinToFahrenheit(Temperature):

assert (Temperature >= 0), “Colder than absolute Zero!”

return ((Temperature-273)\*1.8)+32

print KelvinToFahrenheit(273)

You can also use a try: block to test the code.

Syntax

try:

#insert code

except ExceptionI:

#if there is exception 1, execute this block

except ExceptionII:

#if there is exception 2, execute this block

………………

else:

#if there is no exception, execute this block

**Getting User Input**

Python allows you to read lines of text from keyboard input onto the screen. You can use either function:

raw\_input

Reads one line of standard input and returns it as a string

e.g. str = raw\_input(“Enter your input: “);

Enter your input: Hello Python!

input

Almost the same as raw\_input except it assumes the input is a Python expression and returns an evaluated result.

e.g. str = input(“Enter your input: “);

Enter your input: [x\*5 for x in range(2,10,2)]

**Type Declaration (Extension)**

If you want to define new types for your own use, you can also do so in Python (for those who have programmed in C, C++.)

Noddy Object: A refcount and a pointer to a type object. You can change noddy\_NoddyObject into your own variable name.

e.g.

#include <Python.h>

Typedef struct{

PyObject\_HEAD

/\*Type-specific field”/

} noddy\_NoddyObject;

static PytTypeobject noddy\_NoddyType = {

PyObject\_HEAD\_INIT(NULL)

0 /\*ob\_size\*/

0 /\*tp\_name\*/

0 /\*tp\_basicsize\*/

0 /\*tp\_itemsize\*/

0 /\*tp\_dealloc\*/

0 /\*tp\_print\*/

0 /\*tp\_getattr\*/

0 /\*tp\_setattr\*/

0 /\*tp\_compare\*/

0 /\*tp\_repr\*/

0 /\*tp\_as\_number\*/

0 /\*tp\_as\_sequence\*/

0 /\*tp\_as\_mapping\*/

0 /\*tp\_hash\*/

0 /\*tp\_call\*/

0 /\*tp\_str\*/

0 /\*tp\_getattro\*/

0 /\*tp\_setattro\*/

0 /\*tp\_as\_buffer\*/

Py\_TPFLAGS\_DEFAULT, /\*tp\_flags\*/

“Noddy Objects”, /\*tp\_doc\*/

}

#to enable object creation

PyMODINIT\_FUNC

Initnoddy(void)

{

PyObject\* m;

noddy\_NoddyType.tp\_new = PyType\_GenericNew;

if (PyType\_Ready(&noddy\_Noddy\_Type < 0)

return;

m = Py\_InitModule3(“noddy”, noddy\_methods, “Example module that creates an extension type,”);

Py\_INCREF(&noddy\_NoddyType);

PyModule\_AddObject(m, “Noddy”, (pyObject \*)&noddy\_NoddyType);

}

**Syntax and Semantics**

Syntax

Which strings of characters and symbols are well-formed and valid expressions in programming (legality)

e.g. cat dog boy is not syntactically valid because it is not a form of an acceptable sentence

e.g. 3.2 + 3.2 is a valid Python expression

Semantic

Static: Which syntactically valid strings have a meaning

e.g. I are big is syntactically valid, but not valid English

Meaning associated with the expression (meaning)

e.g. <literal> <operator> <literal> is a valid syntactic form, but 2.3/’abc’ is a static semantic error. Cannot divide number by strings

Formal: What is the meaning associated with a syntactically correct string of symbols with no static semantic errors. Always has exactly one meaning with an expression (meaning + legality)

Checking whether an expression is an integer

if not isinstance(n, int):

#TODO

return None

Accessing an array or list:

arr/list[start:end:steps increment] -> [::] (for the entire list)

converting an array into a string: “joiner”.join(map(str, array))

Splitting a variety of input into a list

input().split() for x in range(n)

Converting the list into a dictionary

dictionary = {k:v for k,v in name}

1. https://docs.python.org/2/howto/unicode.html#python-2-x-s-unicode-support [↑](#footnote-ref-0)