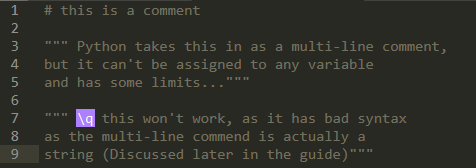
Basics of Python 3

Note: This reference guide has written explanations and screenshots. The screenshots should be sufficient for most sections, but the written explanations are below the screenshots if you need them.

Table of Contents

1. [Installation](#Installation)
2. [IDEs](#IDE)
3. [Comments](#Comments)
4. [Variables](#Variables)
5. [Datatypes](#BasicDatatypes)
6. [Input and Output (I/O)](#InputOutput)
7. [Data Structures](#ComplexDatatypes)
8. [Datatype Conversion](#DatatypeConversion)
9. [Arithmetic operators](#Operators)
10. [Assignment Operators](#assignmentOperators)
11. [Statements](#Statements)
12. [Functions](#Functions)
13. [Indentation and Scope](#Indentation)
14. [Classes](#Classes)
15. [Imports](#Imports)
16. [Inheritance](#Inheritance)
17. [Comparison Operators](#comparisonOperators)
18. [Flow Control](#FlowControl)
19. Installing Python 3
    1. Installer
       1. You can use an installer for Python 3.8.3 from the official Python website.
20. IDEs
    1. If you have a preference, anything that can edit python should be ok.
    2. If you prefer an IDE, Pycharm is free and a recommendation.
       1. With an IDE you can run the programs from the IDE.
    3. If you prefer a text editor, Sublime or Atom are both good recommendations.
    4. Python comes with its own shell and file editor build: IDLE
       1. To use the shell, open IDLE.
       2. To get at the text editor, go to file and create a new file.
21. Comments



Written instructions not interpreted as code. Two types: Single line and Multi line

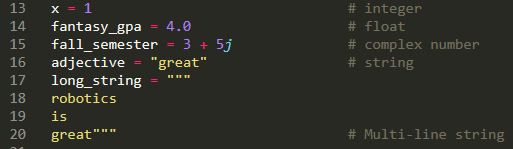
* 1. Single line
     1. Performed by using the # symbol before a statement.
     2. Cannot include illegal characters, ex \q
  2. Multi-Lines
     1. Are actually [strings](#String)
     2. Cannot include illegal string characters, ex \q
        1. Would cause an error.

1. Variables



* 1. Variables
     1. Allocated space for data under a name, ex: Here x is a variable: x = 1
     2. Type-inferred: automatically stored based on how they’re assigned.
        1. No need to explicitly declare type.
        2. Ex: x = 1 will become an [integer](#Integers).

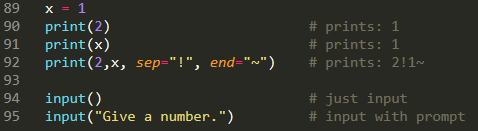
1. Datatypes



The type that is assigned to, for example, a [variable](#Variables).

* 1. Integers
     1. Any variable assigned a whole number, ex: x = 1
  2. Float
     1. Any variable assigned a non-whole number, ex: x = 1.0
  3. Complex number
     1. Any variable assigned a real and imaginary part, ex: x = 1 + 2j
  4. String
     1. Any variable assigned using quotes or apostrophes, ex: x = “robot”
     2. Cannot include illegal characters, ex: \q
     3. Triple quotes or apostrophes indicate a multi-line string.

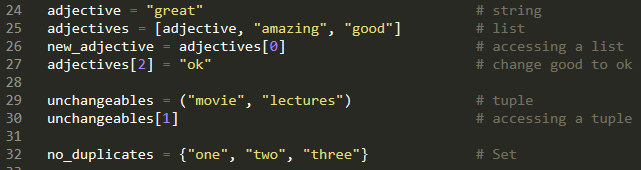
1. Input and Output (I/O)



* 1. Output
     1. Performed using print(outputGoesHere)
        1. Can print variables, some data structures, and strings.
        2. Print multiple things using a comma.
        3. (Optional) Specify a way to end with end=
        4. (Optional) Specify the separator with sep=
        5. (Optional) Specify a file to print to with file=
        6. (Optional) format by specifying an order, {0}, and using .format(x)
  2. Input
     1. Performed using input(promptGoesHere)
        1. Prompt is optional, but is generally a [string](#String).

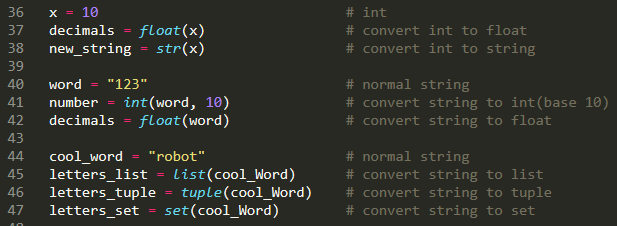
1. Data Structures

These can contain multiple values and hold a collection of items, which can include other data structures.



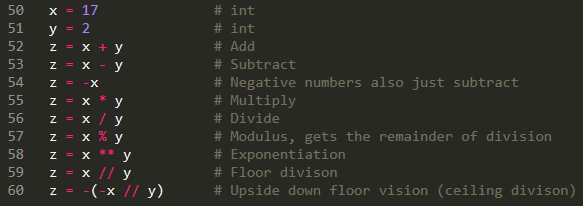
* 1. List
     1. Defined with square brackets, ex: x = [“one”, “two”, “three”]
     2. Ordered, so you can access each part by index.
  2. Tuple
     1. Defined with parenthesis, ex: x = (“one”, “two”, “three”)
     2. Unchangeable. Once something is added, you cannot change it.
  3. Set
     1. Defined with braces, ex: x = {“one”, “two”, “three}
     2. Unordered. You cannot access any part by index.
     3. No duplicates. If you add a duplicate, it will be ignored.

1. Datatype Conversion



* 1. Python will automatically convert when using [operators](#Operators).
  2. Explicit datatype conversion can be done using the conversion operators.
     1. Most of these are intuitively named, ex [str()](#String)

1. Arithmetic Operators



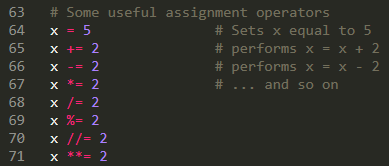
Python includes all the basics, plus modulus, exponents, and floor division.

These can be used with variables or the actual numbers.

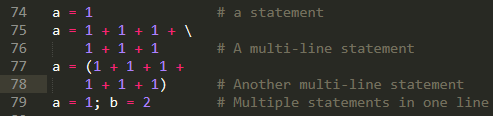
* 1. Modulus
     1. Returns only the remainder of the division.
        1. Ex 17 % 2 will yield 1
  2. Exponents
     1. To the power of.
        1. Ex: x \*\* y is the same as x to the power of y
  3. Floor Division
     1. Rounds down division result to the nearest whole number.
        1. Ex: 17 // 2 will yield 8

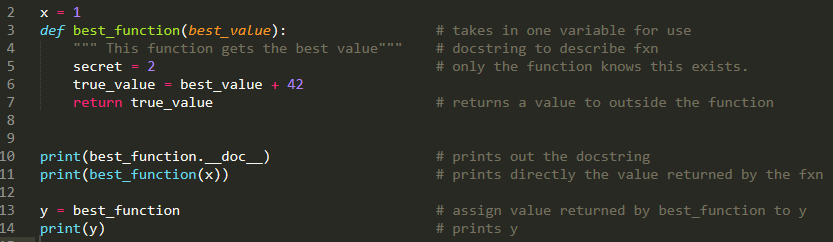
1. Assignment Operators

Used to assign values to variables. Includes all expected, plus [modulus](#Modulus), [exponentiation](#Exponents), and [floor division](#FloorDiv).



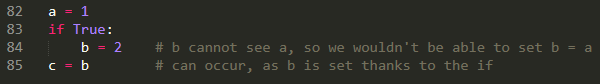
* 1. Outside of =, always can be taken of the form x += 1 is equivalent to x = x + 1

1. Statements
   * 1. Any instruction that can be executed, ex a = 1
     2. You can make a statement multi-line by adding the \ character.
     3. You can extend a statement explicitly with parenthesis, brackets, or braces.
     4. Multiple statement lines are achieved by a ; between statements.
2. Functions



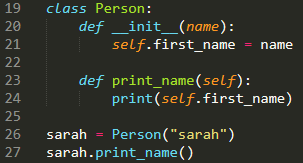
* 1. A group of related statements that perform a task.
  2. Created by using this function header: def name\_of\_function(parameters):
     1. Parameters are just passed by name, as type is inferred.
        1. These are any variables you want to use from outside the function.
     2. Semicolon marks start of the function/block.
  3. You can add a doc string, which is a [multi-line string](#String) after a function header.
     1. This explains what the function does.
  4. To use something from inside the function, outside the function: return the value.
  5. To call a function, use its name + any parameters: name\_of\_functions(words)
     1. If a value is returned, you must set a variable equal to the function.
     2. If the function is in a [class](#Classes), call it with self.name\_of\_function(words)
        1. Additionally, inside a class one parameter must be self.
           1. name\_of\_function(self, words)

1. Indentation and Scope



* 1. Indentation matters!
     1. Defines a block of code. If you’ve used C or Java, indentation is similar to brackets {}.
        1. Python does not support creating a sole block with indentation – it must be used as a body with something like a function, loop, or class.
     2. Anything in the block cannot access outside the block, unless passed in.
     3. Depending on the conditions, things inside the block may be accessed.
        1. Ex, above, it is evaluated as true. Thus, b is set and c can see b.
        2. If it were evaluated as false, b wouldn’t be set. So, c can’t see b.

1. Classes



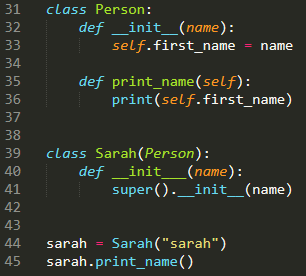
* 1. Conceptually, a gathering of all of the functions and fields needed to create an object.
     1. An object is an instance of a class.
  2. Don’t need to be included in a python module.
  3. Defined as such: class NewClass:
     1. Functions, variables, etc can all be included in here.
     2. A class can have a docstring right below the class name.
        1. A [multi-line string](#String) right after the function header.
  4. Can include an init function, which gives all objects made from the class attributes on creation in accordance with the function.
     1. If unprovided, given a default.
     2. Create as: def \_\_init\_\_(variables):
        1. Add any wanted attributes, such as: self.name = variable
  5. Create an object as such: object = NewClass()
  6. Call functions as such: object.function()
     1. Or: NewClass.function()

1. Imports



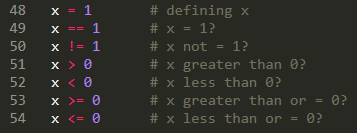
* 1. To import a library/module of useful [functions](#Functions) or [classes](#Classes), use the import keyword.
     1. You can also import the module and rename it.
        1. import real\_name as robots
  2. To import a specific part of a library/module, use: import fxn\_name from module

1. Inheritance



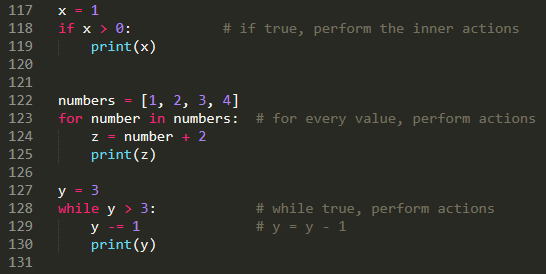
* 1. The ability for one class to inherit all of the methods/variables from another class.
  2. Perform within the class declaration: class Sarah(Person)
     1. Will inherit all aspects of the Person class.

1. Comparison Operators



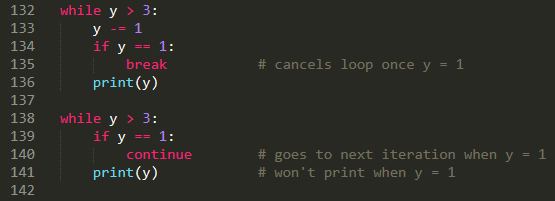
Compare values and return the result of comparisons.

1. Flow Control



Functions that help control the flow of the program.

* 1. if
     1. of form: if condition:
     2. If condition is true, perform what is inside the if.
     3. (Optional) elif (alternate condition) is true, perform what is inside the elif.
     4. (Optional) else perform the action inside the else.
  2. for
     1. of form: for value in sequence:
     2. for each value looped through, perform the inner actions
  3. While
     1. Of form: while condition:
     2. While the condition is true, perform all inner actions



* 1. break and continue
     1. break
        1. breaks out of the loop regardless of condition.
     2. Continue
        1. Continues to the next iteration of the loop