Contents

[1 Some Python related concepts / topics 2](#_Toc528997702)

[1.1 The Python Debugger 2](#_Toc528997703)

[1.2 Everything is an object (mutable or immutable) in Python 2](#_Toc528997704)

[1.2.1 The basics of variable assignments 2](#_Toc528997705)

[1.2.2 Mutability – Imuutability Table 3](#_Toc528997706)

[1.2.3 Id() and Type() functions 3](#_Toc528997707)

[1.2.4 Actual examples with code and diagrams 3](#_Toc528997708)

[1.3 Indentation in Python 4](#_Toc528997709)

[1.4 Iteration in Python 6](#_Toc528997710)

# Some Python related concepts / topics

## The Python Debugger

## Everything is an object (mutable or immutable) in Python

### The basics of variable assignments

When learning Python, one has to quickly understand that is that all objects in Python are either **mutable** or **immutable**.

Everything in Python is an Object, every variable that we create holds an object instance.

* name = “pibm” : <name> = <object>
* Val = 1 : <name> = <object>
* myList = [1,2,3] : <name> = <object>

When an object is created *(Other terms used – “instantiated”, “initialized”)*, it is assigned a unique object id.

Its type is defined at runtime and once set can never change, however its state can be changed if it is mutable.

In short, the state (“values”) of a **mutable** object can be changed after it is created, and an **immutable** object’s state (“values”) can’t.

When we assign a value to a variable :

We are actually **binding** a **name** to an **object.**One implication of this is that multiple names can be bound to a single object. (Multiple labels can be assigned to the same object).

The variable is actually a label that we assign to an object, it is the way we, as developers can identify it. However, what is always important about the underlying object is its value (“state”) and its type.

### Mutability – Imuutability Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Description** | **Immutable** |  |
| bool | Boolean Value | YES |  |
| int | Integers | YES |  |
| float | Floating point numbers | YES |  |
| List | Mutable sequence of objects | NO |  |
| tuple | Immutable sequence of objects | YES |  |
| Str | Character string | YES |  |
| set | Unordered set of distinct objects | NO |  |
| frozenset | Immutable form of “set” class | YES |  |
| dict | Key value pairs  (associative mappings) | NO |  |

### Id() and Type() functions

* Id() - returns the actual memory location where the variable is stored
* type() – returns the type of the object that the variable is bound to

### Actual examples with code and diagrams

* Example 001

|  |
| --- |
| #py\_everything\_is\_an\_obj\_001.py  a = "spam"  b = "spam"  #  print(id(a))  print(id(b))  #  # id() returns the actual memory location where the variable is stored.  # Since id(a) = id(b), we know that a and b both point to a single variable,  # that resides in a single memory location.  # This is what we mean by “multiple names bound to single object  print(a is b) |

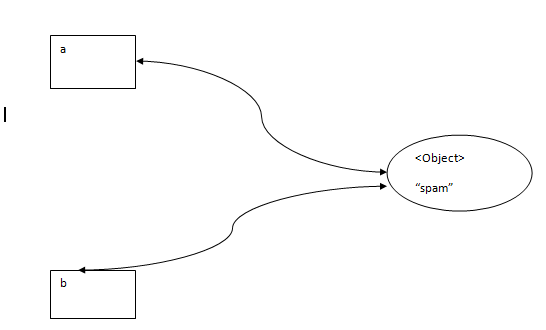
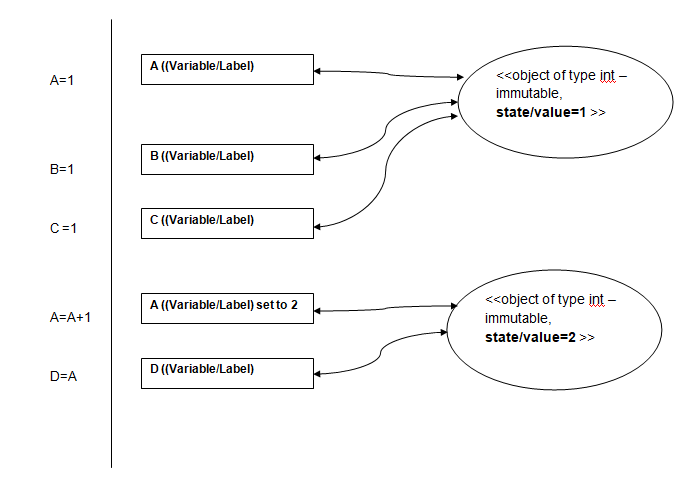


Figure –

* Example 002



## Indentation in Python

### Sample Images

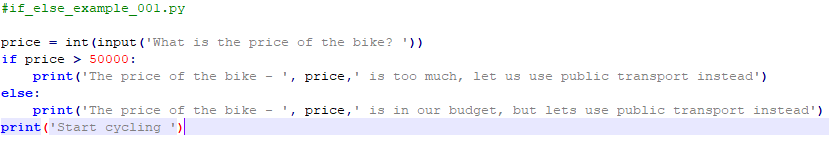


Figure –

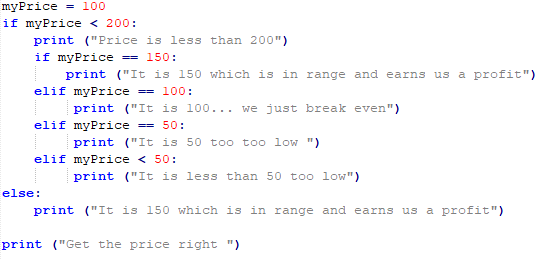
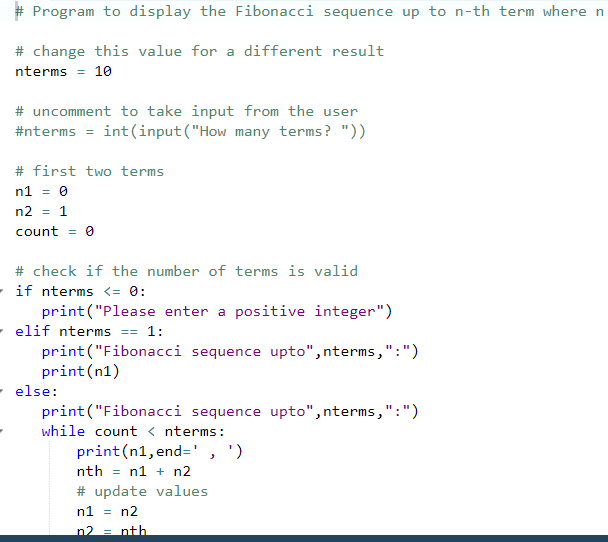


Figure -



How many spaces should you indent? Python requires at least one, some programmers consistently use two, four is the most popular number, but some prefer a more dramatic display and use eight. A four space indentation for a block is the recommended Python style. This text uses the recommended four spaces to set off each enclosed block. In most programming editors you can set the Tab key to insert spaces automatically so you need not count the spaces as you type. Whichever indent distance you choose, you must use this same distance consistently throughout a Python program.

Why is indentation that mixes tabs and spaces a problem and thus forbidden in Python 3? Consider creating a Python source file in one editor and then viewing it in a different editor with tab stops set differently. Lines that appear perfectly indented in the original editor would be misaligned in the new editor. Instead, code indented with four spaces within one editor would appear exactly the same in any other editor. Python 3 does allow the use of tabs for indentation—you just cannot mix them with spaces within the same source file.

## Iteration in Python