Important Python Concepts – in Detail

-- Arunkumar Nair

04 Jan 2020

Data Types, Data Structures

Help for Python

More Information about Python

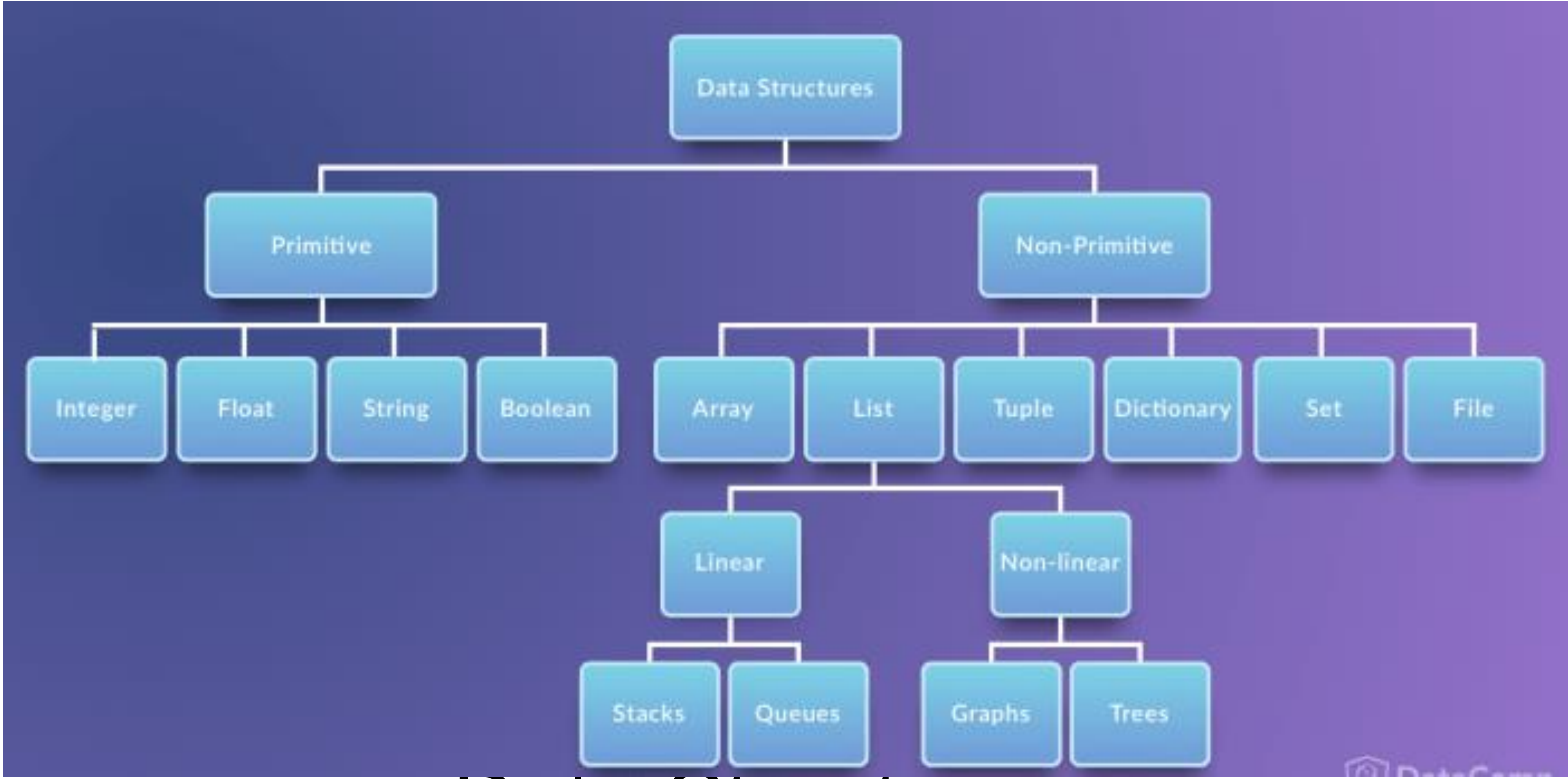
https://docs.python.org/3/

More information about data types

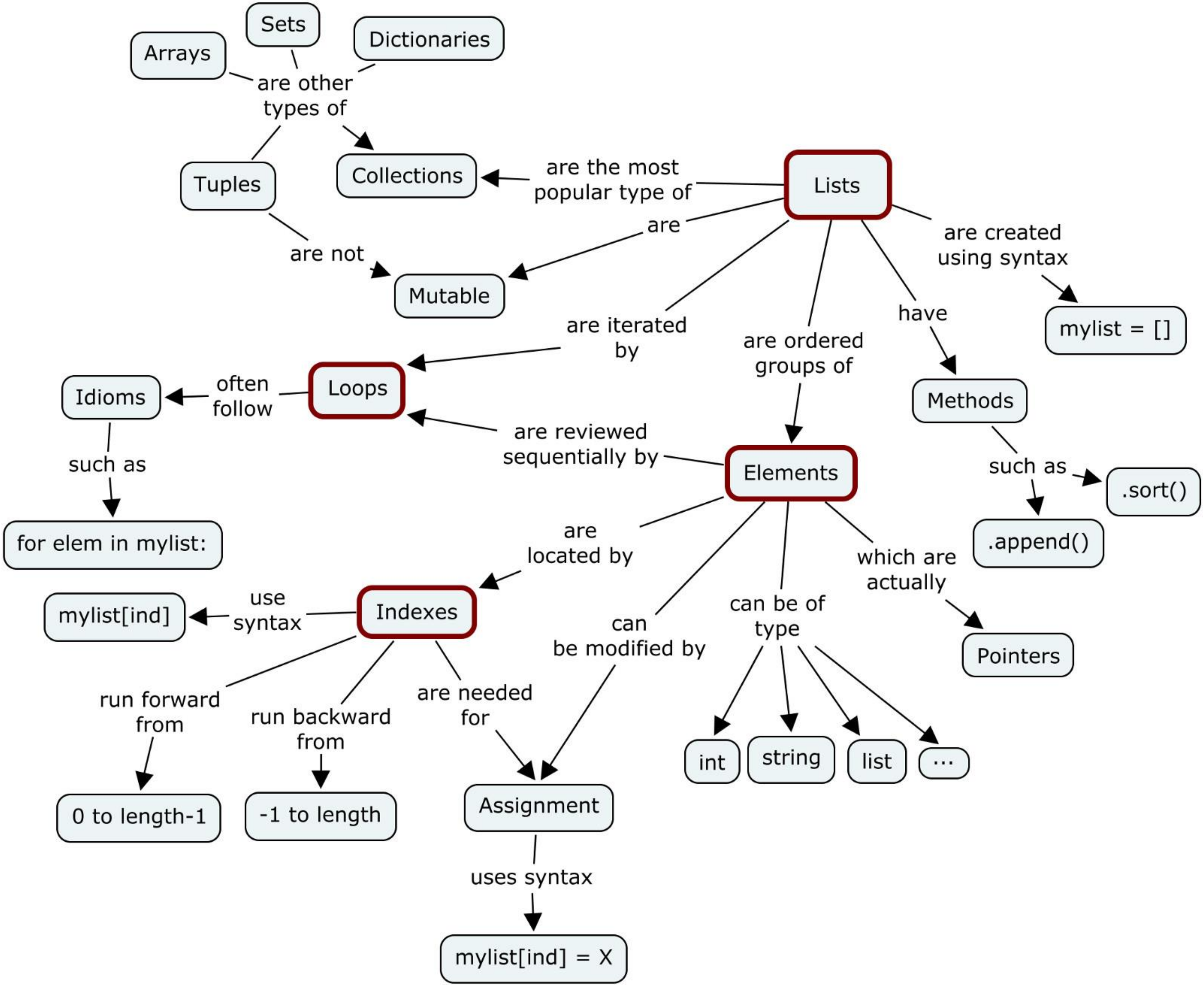
https://docs.python.org/3/library/datatypes.html

Immutable:
In primitive type, it copies the actual value and therefore changing one value won't change a different variable even if they were used the same root var..
In Python : Tuple is Immutable

Mutable
Non primitive however, only use references and therefore changing one value will change in all the vars that store the same reference...



Data Structures



Sets

Arrays

Dictionaries

are other types of

Tuples

Collections

are the most popular type of

Lists

are

are not

Mutable

are created using syntax

mylist = []

are iterated by

are ordered groups of

have

Loops

often follow

Idioms

Methods

are reviewed sequentially by

Elements

such as

such as

.sort()

for elem in mylist:

.append()

are located by

which are actually

Indexes

use syntax

mylist[ind]

can be modified by

can be of type

Pointers

run forward from

run backward from

are needed for

int

string

list

...

0 to length-1

-1 to length

Assignment

uses syntax

mylist[ind] = X

Tuples are Immutable ?

Python tuples have a surprising trait: they are immutable, but their values may change.

This may happen when a tuple holds a reference to any mutable object, such as a list.

Variables are NOT like boxes where you store data.

http://radar.oreilly.com/2014/10/python-tuples-immutable-but-potentially-changing.html

Mutable and Immutable

https://medium.com/@meghamohan/mutable-and-immutable-side-of-python-c2145cf72747

Tuples are Immutable?

What is immutable is the physical content of a tuple, consisting of the object references only. The value of the list referenced by dum[1] changed, but the referenced object id is still the same. A tuple has no way of preventing changes to the values of its items, which are independent objects and may be reached through references outside of the tuple, like the skills name we used earlier. Lists and other mutable objects inside tuples may change, but their ids will always be the same.

Types of Data Structures

- String
- Data Structures
  - List
  - Tuple
  - Set
  - Dictionary

String

- Its a sequence of characters
- Single Quotes & Double Quotes are treated similar
- Triple Quotes : doc-string, multiline comments
- Strings are immutable
- Representation & Slicing

String

- Methods :-
  - count
  - index
  - find
  - join
  - Isalnum, isalpha, isdigit, islower, isspace, isupper
  - lstrip, rstrip, replace, partition, split, rsplit
  - endswith, startswith

List

- Ordered collections of heterogeneous objects
- Representation is similar to String
- Mutable, can grow and shrink whenever required

List

- Methods :-
  - count
  - append
  - extend
  - index
  - insert, pop, remove, reverse, sort
  - __add__, __contains__, __eq__, __ge__, __sizeof__ .....

Tuple

- Ordered collections of heterogenous objects
- Representation is similar to String, List
- Immutable
- Methods :-
  - count
  - index
  - __mul__, __add__, __contains__, __eq__, __ge__, __sizeof__ .....

Set

- Ordered collections of heterogeneous unique objects
- Representation is similar to String, List but cannot be sliced or accessed using index
- Mutable

Set

- Methods :-
  - add
  - pop
  - clear
  - copy
  - discard v/s remove (raise KeyError)
  - difference, intersection, union .....

Dictionary

- Unordered collection of key-value pairs
- Representation is similar to String, List but cannot be sliced or accessed using index.
- Mutable

Dictionary

- Methods :-
  - copy
  - fromkeys
  - has_key
  - items
  - keys, values
  - iteritems, iterkeys, itervalues
  - pop, popitem
  - viewitems, viewkeys, viewvalues

Slicing

Slicing : extracting specific set of data from the container.

Lists, Strings & Tuple support Slicing

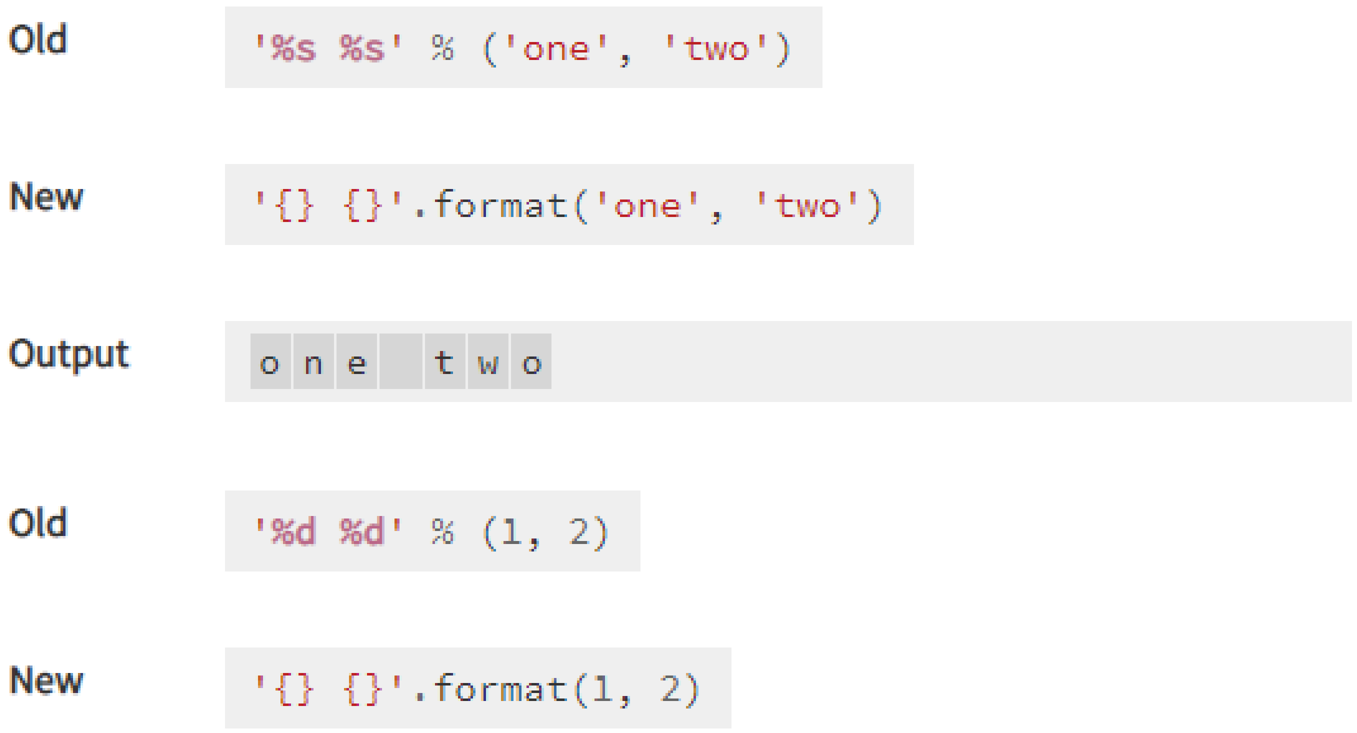
Reverse Indexing

Formatting for Printing

Format

Basic Formatting-Simple positional formatting is probably the most common use-case

https://docs.python.org/3.4/library/string.html



Old

```
'%s %s' % ('one', 'two')
```

New

```
'{} {}'.format('one', 'two')
```

Output

```
one two
```

Old

```
'%d %d' % (1, 2)
```

New

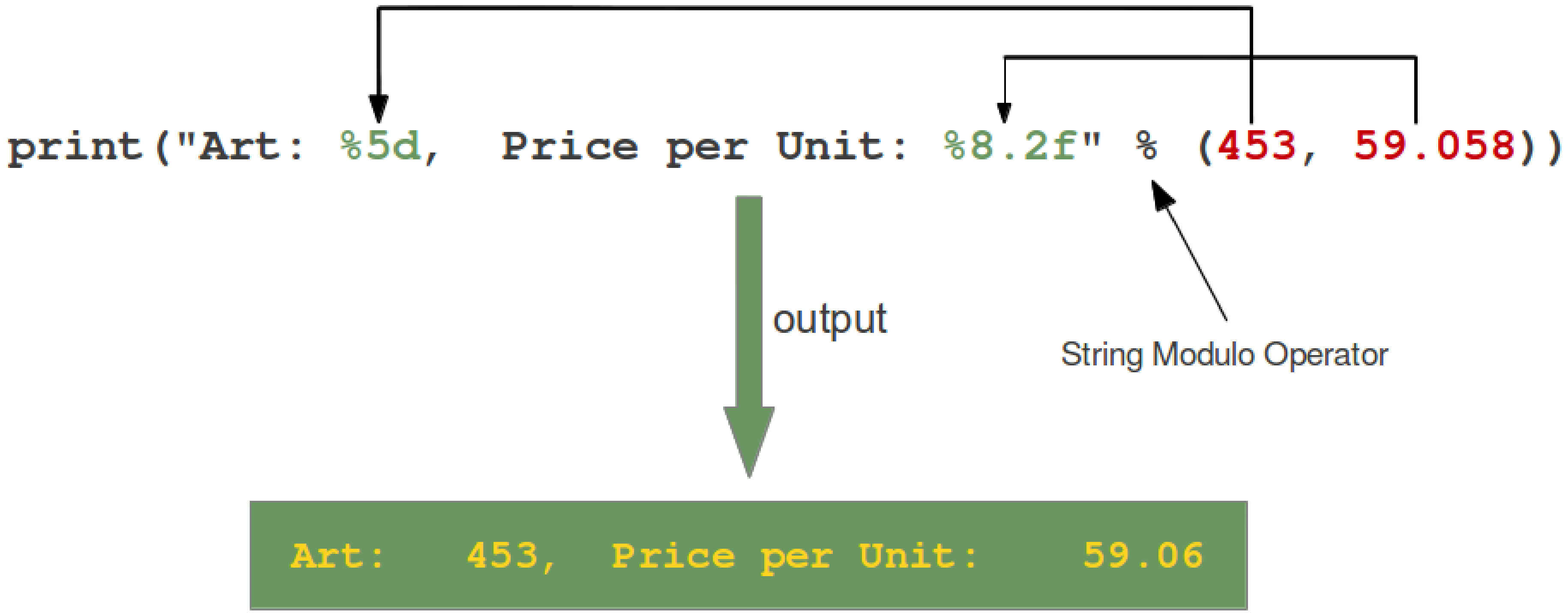
```
'{} {}'.format(1, 2)
```

Format

How % string modulo operator works

https://www.python-course.eu/python3_formatted_output.php

(Below is the old style)



```
print("Art: %5d, Price per Unit: %8.2f" % (453, 59.058))
```

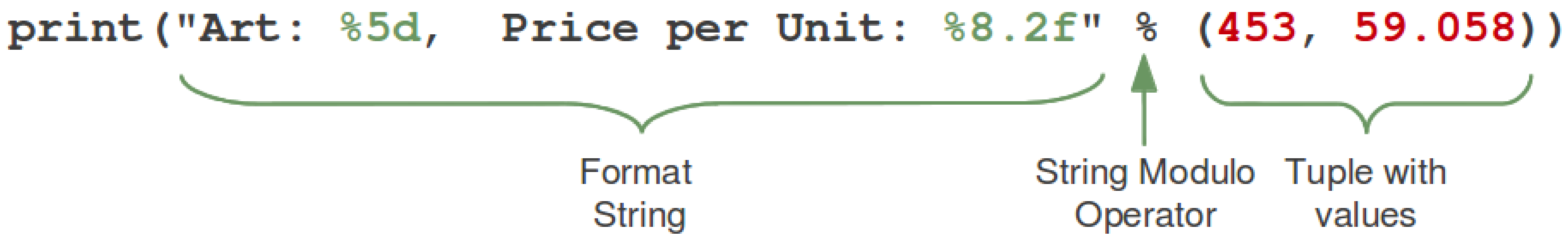
```
Art:   453, Price per Unit:    59.06
```

Format

How % string modulo operator works

https://www.python-course.eu/python3_formatted_output.php

(Below is the old style)



```
print("Art: %5d,  Price per Unit: %8.2f" % (453, 59.058))
```

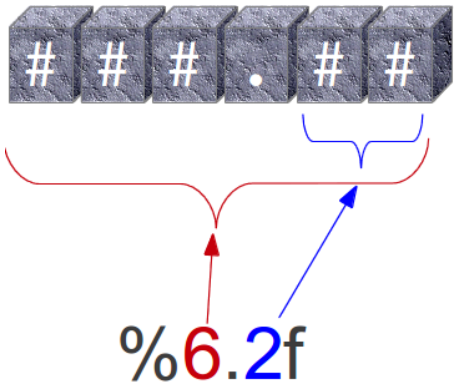
Format

How % string modulo operator works

https://www.python-course.eu/python3_formatted_output.php

(Below is the old style)

The second one "%8.2f" is a format description for a float number. Like other placeholders, it is introduced with the "%" character. This is followed by the total number of digits the string should contain. This number includes the decimal point and all the digits, i.e. before and after the decimal point. Our float number 59.058 has to be formatted with 8 characters. The decimal part of the number or the precision is set to 2, i.e. the number following the "." in our placeholder. Finally, the last character "f" of our placeholder stands for "float".



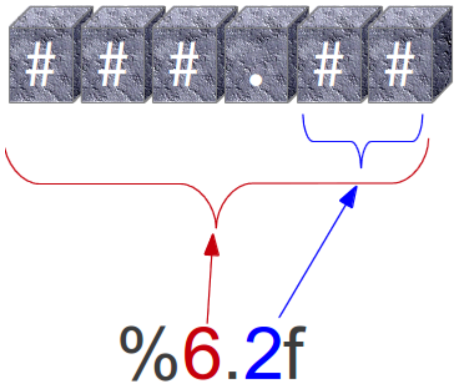
Format

How % string modulo operator works

https://www.python-course.eu/python3_formatted_output.php

(Below is the old style)

The second one "%8.2f" is a format description for a float number. Like other placeholders, it is introduced with the "%" character. This is followed by the total number of digits the string should contain. This number includes the decimal point and all the digits, i.e. before and after the decimal point. Our float number 59.058 has to be formatted with 8 characters. The decimal part of the number or the precision is set to 2, i.e. the number following the "." in our placeholder. Finally, the last character "f" of our placeholder stands for "float".



Format

NEW STYLE of Positional Parameter of the format {}

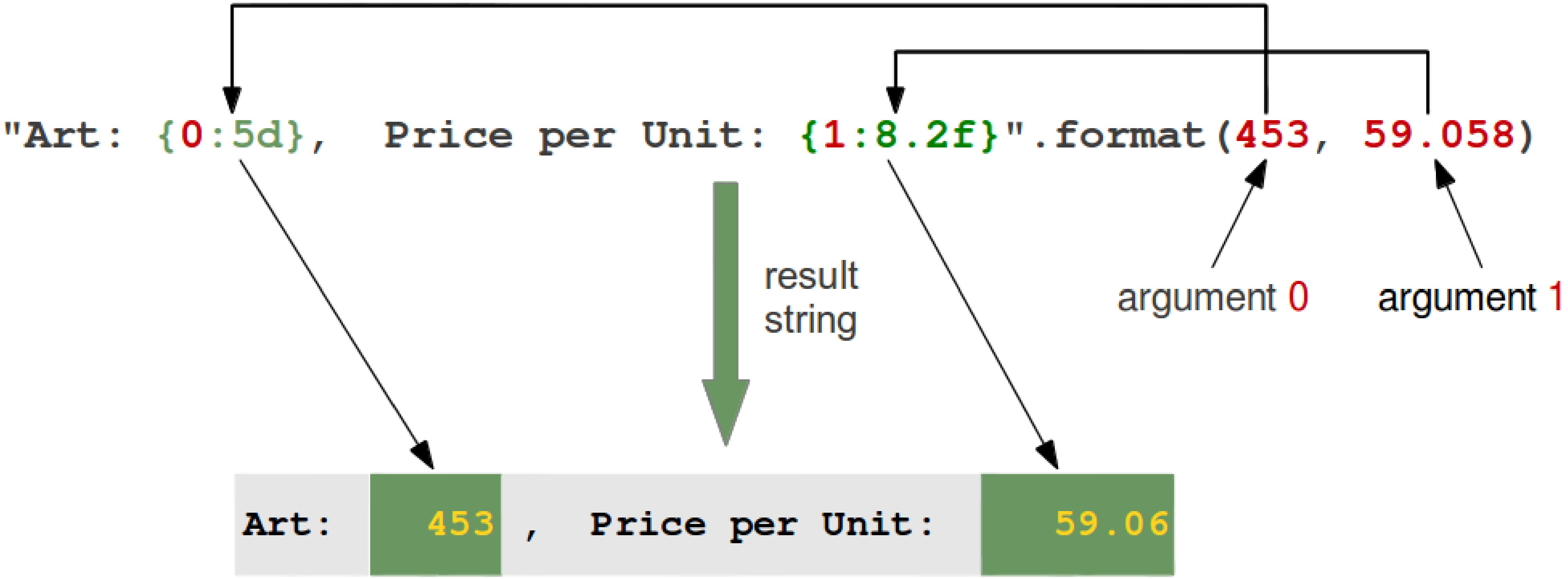
https://www.python-course.eu/python3_formatted_output.php

A positional parameter of the format method can be accessed by placing the index of the parameter after the opening brace, e.g. {0} accesses the first parameter, {1} the second one and so on. The index inside of the curly braces can be followed by a colon and a format string, which is similar to the notation of the string modulo, which we had discussed in the beginning of the chapter of our tutorial, e.g. {0:5d}
If the positional parameters are used in the order in which they are written, the positional argument specifiers inside of the braces can be omitted, so '{} {} {}' corresponds to '{0} {1} {2}'. But they are needed, if you want to access them in different orders: '{2} {1} {0}‘

Format

NEW STYLE of Positional Parameter of the format {}

https://www.python-course.eu/python3_formatted_output.php



```
"Art: {0:5d}, Price per Unit: {1:8.2f}".format(453, 59.058)
```

result
string

argument 0

argument 1

```
Art:   453 , Price per Unit:    59.06
```

Format

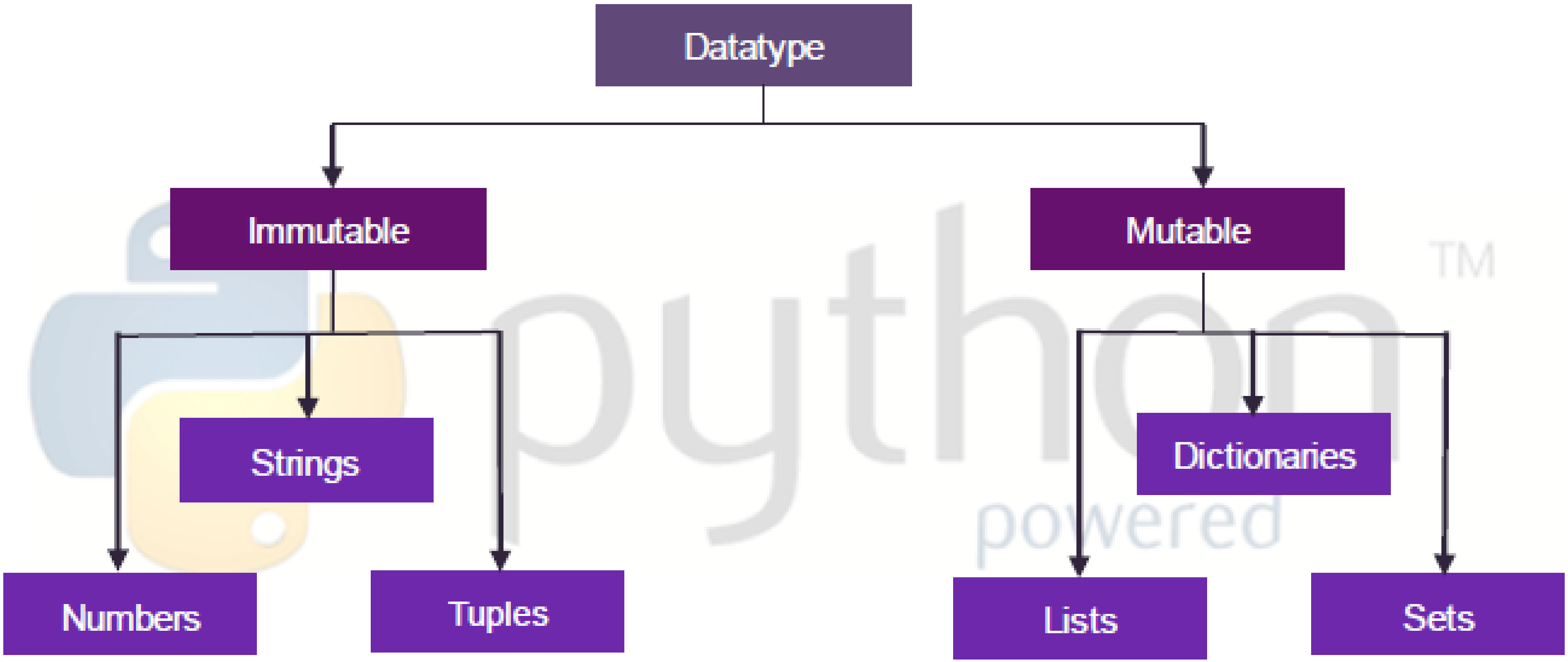
Join()

https://www.python-course.eu/python3_formatted_output.php

join() function in Python

The join() method is a string method and returns a string in which the elements of sequence have been joined by str separator.

Immutable Vs Mutable



Datatype

Immutable

Mutable

Strings

Numbers

Tuples

Dictionaries

Lists

Sets

Mutable vs Immutable Example

***Mutable** instances is passed by reference.*

***Immutable** instances is passed by value.*

Abstract example. Suppose that exist a file named *txtfile* in my HDD. Now, when you ask *txtfile* from me, I can return it in two modes:

1. **Create a shortcut to *txtfile* and pas shortcut to you** (Mutable)
2. **Take a copy for *txtfile* and pas copy to you.(**Immutable)

Mutable vs Immutable Example

1. **Create a shortcut to *txtfile* and pas shortcut to you** (Mutable)
   In first mode, returned *txtfile* is a mutable file, because when you do changes in shortcut file, you do changes in original file too. Advantage of this mode is that each returned shortcut required less memory (on RAM or in HDD) and disadvantage is that everyone (not only me, owner) have permissions to modify file content.

Mutable vs Immutable Example

2. **Take a copy for *txtfile* and pas copy to you.(**Immutable)

In second mode, returned *txtfile* is an immutable file, because all changes in received file does not refer to the original file. Advantage of this mode is that only me (owner) can modify original file and disadvantage is that each returned copy required memory (in RAM or in HDD).

By Val, By Ref

Different way in which ByVal and ByRef are implemented in Python

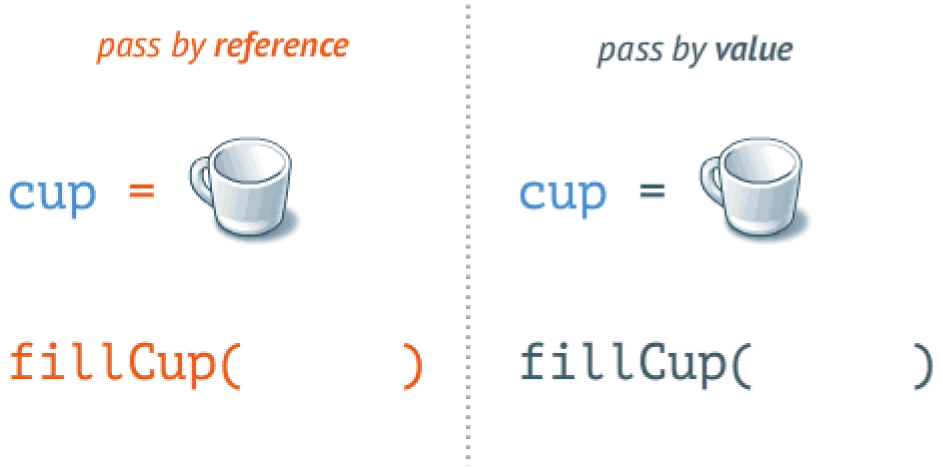
(Python has no ByVal and ByRef Implementation, there are only work around)

Object references are passed by value

Amongst programming languages are pass-by-reference and pass-by-value. Unfortunately, Python is “pass-by-object-reference”, of which it is often said:

“Object references are passed by value.”

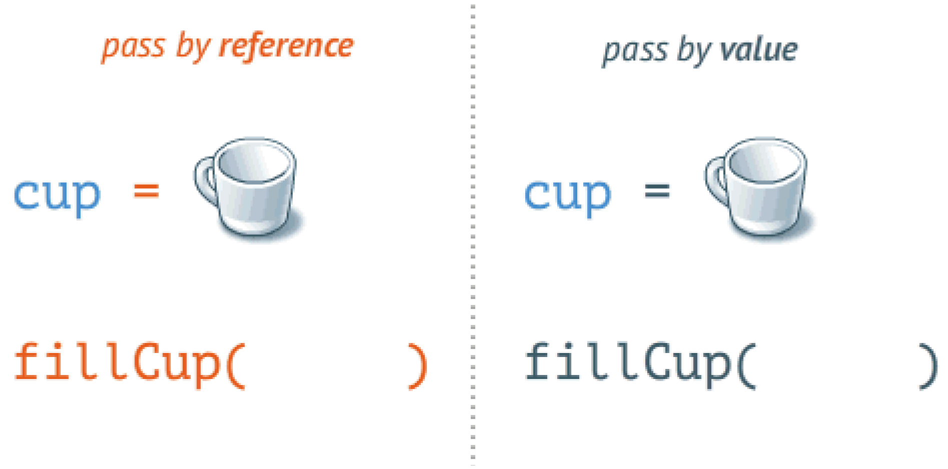
Pass by Ref and Pass by Value



By Reference
A also changes
It maintains a Reference to the location

By Value
A doesn’t change
Because it maintains a copy

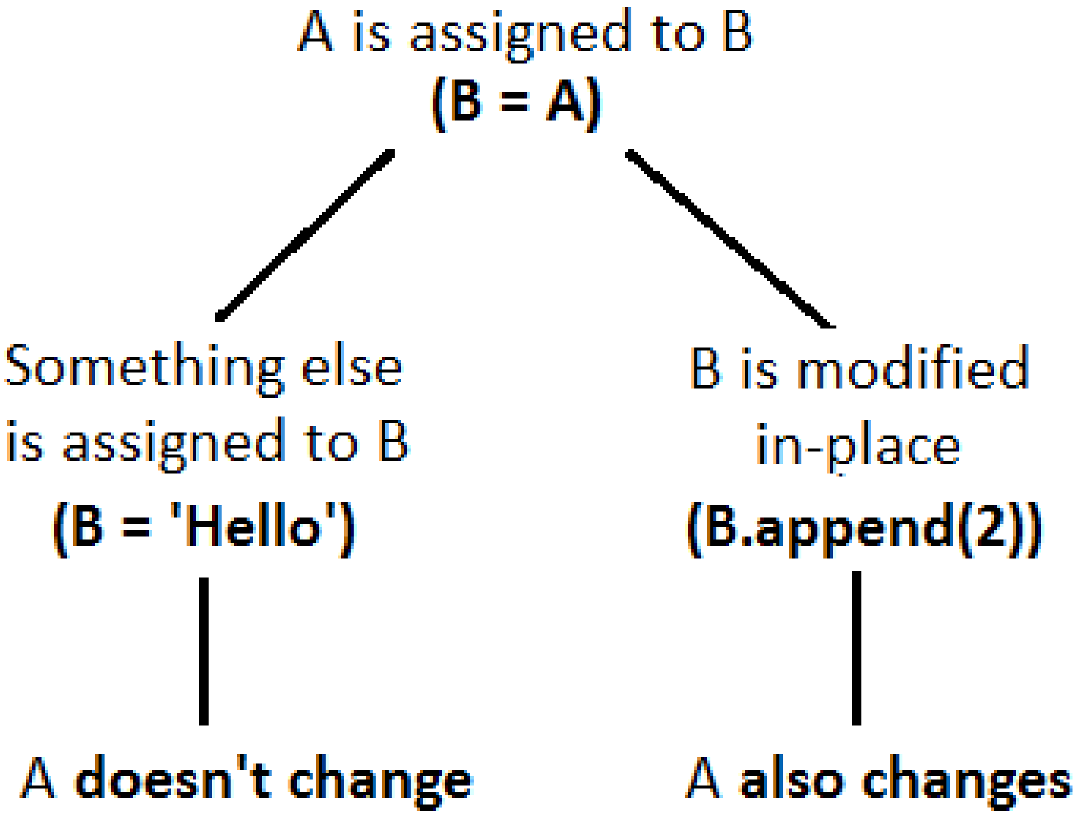
Pass by Ref and Pass by Value



Pass by reference means that a number (called a memory address) is passed on, this address defines where the value is stored.

pass by value means the actual value is passed on.

ByVal vs ByRef



A is assigned to B
(B = A)

Something else
is assigned to B
(B = 'Hello')

B is modified
in-place
(B.append(2))

A **doesn't change**

A **also changes**

By Value
A doesn’t change
Because it maintains a copy

By Reference
A also changes
It maintains a Reference to the location

Mutable object : By Reference

If you pass a *mutable* object into a method, the method gets a reference to that same object and you can mutate it to your heart's delight, but if you rebind the reference in the method, the outer scope will know nothing about it, and after you're done, the outer reference will still point at the original object.

Immutable object : By Value

If you pass an *immutable* object to a method, you still can't rebind the outer reference, and you can't even mutate the object.

There are techniques to change the values of Immutable by using the RETURN VALUE

End