

Hi! I am Python



Python Programming Language

Hi I am Python
Programming Language



Table Of Contents



Data Types



Memory Management in Python



Problem Solving

Variables in Memory

a = 10
memory for 10 is 12345

b = 10
memory for 10 is 12345

a = 12
memory for 12 is 12347

b = 11
memory for 11 is 12346

Data Types

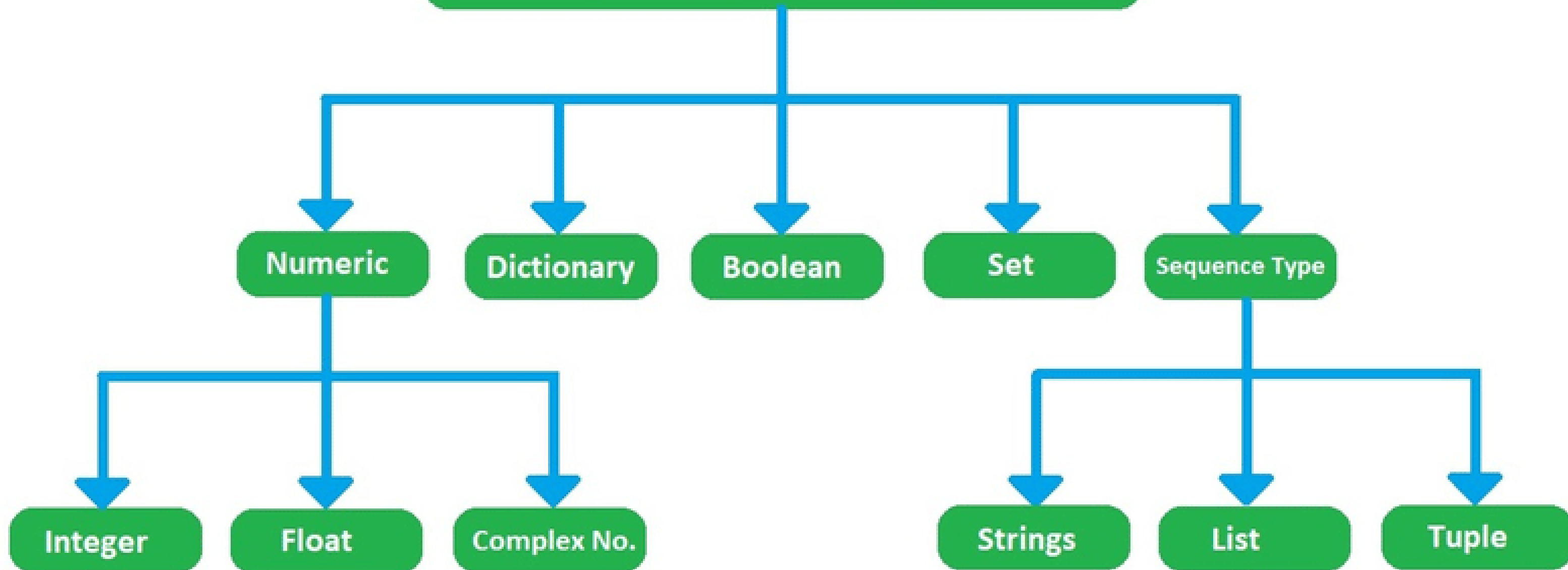
Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, **data types are actually classes and variables are instance (object) of these classes.**



Built in Data Types(cont.)

www.TheEngineeringProjects.com

Python DataTypes



Numeric Data

numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

- **Integers** – This value is represented by int class. **In Python there is no limit to how long an integer value can be.** -3,-2,-1,0,1,2,3 . **int()**
- **Float** – This value is represented by float class. It is a real number with floating point representation. 9.34,9.00,3.45 . **float()**
- **Complex Numbers** – Complex number is represented by complex class. It is specified as (real part) + (imaginary part)j. For example – **2+3j . complex()**

Numeric Data type example using type() function :

type checking :

```
>>> a = 5
>>> type(a)
<class 'int'>
>>> b = 5.6
>>> type(b)
<class 'float'>
>>> c = 3+4j
>>> type(c)
<class 'complex'>
>>> |
```

type conversion :

```
>>> int(5.6)
5
>>> float(5)
5.0
>>> complex(5,4)
(5+4j)
>>> int(5+4j)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    int(5+4j)
TypeError: can't convert complex to int
>>> float(5+4j)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    float(5+4j)
TypeError: can't convert complex to float
>>> complex(5)
(5+0j)
>>> complex(5.6)
(5.6+0j)
```


Sequence Type

In Python, sequence is the **ordered collection of similar or different data types**.

1) String : (**immutable**)

A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by **str** class. example : **'hello', "hello", """"hello""". str()**

2) List : (**mutable**)

Lists are just like the arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type . **example : data = [1,2,3,4,5,'food','fruits', 4,5,'food']. list()**

String

Method	Description
<u>capitalize()</u>	Converts the first character to upper case
<u>casefold()</u>	Converts string into lower case
<u>center()</u>	Returns a centered string
<u>count()</u>	Returns the number of times a specified value occurs in a string
<u>encode()</u>	Returns an encoded version of the string
<u>endswith()</u>	Returns true if the string ends with the specified value
<u>expandtabs()</u>	Sets the tab size of the string
<u>find()</u>	Searches the string for a specified value and returns the position of where it was found
<u>format()</u>	Formats specified values in a string
<u>format_map()</u>	Formats specified values in a string
<u>index()</u>	Searches the string for a specified value and returns the position of where it was found
<u>isalnum()</u>	Returns True if all characters in the string are alphanumeric
<u>isalpha()</u>	Returns True if all characters in the string are in the alphabet
<u>isascii()</u>	Returns True if all characters in the string are ascii characters
<u>isdecimal()</u>	Returns True if all characters in the string are decimals
<u>isdigit()</u>	Returns True if all characters in the string are digits
<u>isidentifier()</u>	Returns True if the string is an identifier

String

<u>islower()</u>	Returns True if all characters in the string are lower case
<u>isnumeric()</u>	Returns True if all characters in the string are numeric
<u>isprintable()</u>	Returns True if all characters in the string are printable
<u>isspace()</u>	Returns True if all characters in the string are whitespaces
<u>istitle()</u>	Returns True if the string follows the rules of a title
<u>isupper()</u>	Returns True if all characters in the string are upper case
<u>join()</u>	Converts the elements of an iterable into a string
<u>ljust()</u>	Returns a left justified version of the string
<u>lower()</u>	Converts a string into lower case
<u>lstrip()</u>	Returns a left trim version of the string
<u>maketrans()</u>	Returns a translation table to be used in translations
<u>partition()</u>	Returns a tuple where the string is parted into three parts
<u>replace()</u>	Returns a string where a specified value is replaced with a specified value
<u>rfind()</u>	Searches the string for a specified value and returns the last position of where it was found
<u>rindex()</u>	Searches the string for a specified value and returns the last position of where it was found
<u>rjust()</u>	Returns a right justified version of the string
<u>rpartition()</u>	Returns a tuple where the string is parted into three parts
<u>rsplit()</u>	Splits the string at the specified separator, and returns a list
<u>rstrip()</u>	Returns a right trim version of the string

String

<code>split()</code>	Splits the string at the specified separator, and returns a list
<code>splitlines()</code>	Splits the string at line breaks and returns a list
<code>startswith()</code>	Returns true if the string starts with the specified value
<code>strip()</code>	Returns a trimmed version of the string
<code>swapcase()</code>	Swaps cases, lower case becomes upper case and vice versa
<code>title()</code>	Converts the first character of each word to upper case
<code>translate()</code>	Returns a translated string
<code>upper()</code>	Converts a string into upper case
<code>zfill()</code>	Fills the string with a specified number of 0 values at the beginning

Creating List

Creating a List

```
List = []
```

#creating list using list() method/function:

```
List = list('hello') → ['h', 'e', 'l', 'l', 'o']
```

Creating a List with the use of a String

```
List = ['workingwithlist']
```

Creating a List with the use of multiple values

```
List = ["working", "with", "list"]
```

Creating a Multi-Dimensional List (By Nesting a list inside a List)

```
List = [['working', 'with'], ['list']]
```

List Methods

Method	Description
<u>append()</u>	Adds an element at the end of the list
<u>clear()</u>	Removes all the elements from the list
<u>copy()</u>	Returns a copy of the list
<u>count()</u>	Returns the number of elements with the specified value
<u>extend()</u>	Add the elements of a list (or any iterable), to the end of the current list
<u>index()</u>	Returns the index of the first element with the specified value
<u>insert()</u>	Adds an element at the specified position
<u>pop()</u>	Removes the element at the specified position
<u>remove()</u>	Removes the first item with the specified value
<u>reverse()</u>	Reverses the order of the list
<u>sort()</u>	Sorts the list

Creating Tuple

3) Tuple : (immutable)

Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are **immutable** i.e. **tuples cannot be modified after it is created**. It is represented by tuple class. tuple()
example : (1,2,3,4,'food', 4,5)

Creating Tuple

Creating an empty tuple

```
Tuple1 = ()
```

Creating a Tuple with the use of Strings

```
Tuple1 = ('coding', 'For')
```

Creating a Tuple with the use of list

```
list1 = [1, 2, 4, 5, 6]
```

```
tup = tuple([1,2,3,4,5,6]) → (1, 2, 4, 5, 6)
```

Creating a Tuple with the use of built-in function

```
Tuple1 = tuple('python') → ('p', 'y', 't', 'h', 'o', 'n')
```

Creating a Tuple with nested tuples

```
Tuple1 = (0, 1, 2, 3)
```

```
Tuple2 = ('python', 'geek')
```

```
Tuple3 = (Tuple1, Tuple2)
```


Type Conversion

```
>>> a = 'jack'
>>> list(a)
['j', 'a', 'c', 'k']
>>> tuple(a)
('j', 'a', 'c', 'k')
>>> b = list(a)
>>> b
['j', 'a', 'c', 'k']
>>> c = tuple(b)
>>> c
('j', 'a', 'c', 'k')
>>> d = list(c)
>>> d
['j', 'a', 'c', 'k']
>>> type(a)
<class 'str'>
>>> type(b)
<class 'list'>
>>> type(c)
<class 'tuple'>
>>> type(d)
<class 'list'>
```

indexing

Indexing in Python is a way to refer the individual items within an iterable by its position. In other words, you can directly access your elements of choice within an iterable and do various operations depending on your needs.

G	E	E	K	S	F	O	R	G	E	E	K	S
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

indexing

```
>>> a = 'hello'
>>> a[0]
'h'
>>> a[1]
'e'
>>> a[7]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a[7]
IndexError: string index out of range
>>> b = [1,2,3,4]
>>> b[2]
3
>>> c = (1,2,3,4)
>>> c[3]
4
>>> a[0]='H'
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    a[0]='H'
TypeError: 'str' object does not support item assignment
>>> b[0]=10
>>> b
[10, 2, 3, 4]
>>> c[0]=10
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    c[0]=10
TypeError: 'tuple' object does not support item assignment
>>> |
```

Set

Set is an **unordered** and **unindexed** collection of items in Python. **iterable, mutable and has no duplicate elements**. Unordered means when we display the elements of a set, it will come out in a random order. Unindexed means, we cannot access the elements of a set using the indexes like we can do in list and tuples.

example : {1,2,3,4}

Set

Creating a Set

```
set1 = set()
```

```
print("Initial blank Set: ", set1) → Initial blank Set: set()
```

Creating a Set with the use of a String

```
set1 = set("phiiitrrroonn") → {'p', 'h', 't', 'i', 'r', 'o', 'n'}
```

Creating a Set with the use of a List

```
set1 = set(["learn", "and", "learn"]) → {'learn', 'and'}
```

Creating a Set with a mixed type of values (Having numbers and strings)

```
set1 = set([1, 2, 'hello', 4, 'world', 6, 'hello', 'Hello']) → {'hello', 1, 2, 4, 6, 'world', 'Hello'}
```

Set

Method	Description
<u>add()</u>	Adds an element to the set
<u>clear()</u>	Removes all the elements from the set
<u>copy()</u>	Returns a copy of the set
<u>difference()</u>	Returns a set containing the difference between two or more sets
<u>difference_update()</u>	Removes the items in this set that are also included in another, specified set
<u>discard()</u>	Remove the specified item
<u>intersection()</u>	Returns a set, that is the intersection of two or more sets
<u>intersection_update()</u>	Removes the items in this set that are not present in other, specified set(s)
<u>isdisjoint()</u>	Returns whether two sets have a intersection or not
<u>issubset()</u>	Returns whether another set contains this set or not
<u>issuperset()</u>	Returns whether this set contains another set or not
<u>pop()</u>	Removes an element from the set
<u>remove()</u>	Removes the specified element
<u>symmetric_difference()</u>	Returns a set with the symmetric differences of two sets
<u>symmetric_difference_update()</u>	inserts the symmetric differences from this set and another
<u>union()</u>	Return a set containing the union of sets
<u>update()</u>	Update the set with another set, or any other iterable

Dictionary

Dictionary in Python is an **unordered collection** of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

{1: 'Learn', 2: 'with', 3: 'encodemy'}

- **no duplicate key is allowed**
- **The values in the dictionary can be of any type, while the keys must be immutable like numbers, tuples, or strings.**
- **Dictionary keys are case sensitive- Same key name but with the different cases are treated as different keys in Python dictionaries.**

Creating Dictionary

Creating an empty Dictionary

```
Dict = {}
```

Creating a Dictionary with Integer Keys

```
Dict = {1: 'Learn', 2: 'with', 3: 'phitron'}
```

Creating a Dictionary with Mixed keys

```
Dict = {'Name': 'phitron', 1: [1, 2, 3, 4]}
```

Creating a Dictionary with dict() method

```
Dict = dict({1: 'Learn', 2: 'with', 3: 'phitron'})
```

Creating a Dictionary with each item as a Pair

```
Dict = dict([(1, 'phitron'), (2, 'with')])
```

#keys must be unique

```
members = {1001: "John", 1002: "Jane", 1003: "Emily", 1001: "Max"}print(members)  
{1001: 'Max', 1002: 'Jane', 1003: 'Emily'}
```


Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
<code>clear()</code>	Removes all the elements from the dictionary
<code>copy()</code>	Returns a copy of the dictionary
<code>fromkeys()</code>	Returns a dictionary with the specified keys and value
<code>get()</code>	Returns the value of the specified key
<code>items()</code>	Returns a list containing a tuple for each key value pair
<code>keys()</code>	Returns a list containing the dictionary's keys
<code>pop()</code>	Removes the element with the specified key
<code>popitem()</code>	Removes the last inserted key-value pair
<code>setdefault()</code>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<code>update()</code>	Updates the dictionary with the specified key-value pairs
<code>values()</code>	Returns a list of all the values in the dictionary

Summary

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	[] or list()	[5.7, 4, 'yes', 5.7]
Tuple	Yes	No	() or tuple()	(5.7, 4, 'yes', 5.7)
Set	No	Yes	{ }* or set()	{5.7, 4, 'yes'}
Dictionary	No	No**	{ } or dict()	{'Jun': 75, 'Jul': 89}

Thank You

