Python - Data model

Data types are the fundmental elements of any programming language. Python has different data types - simple and advanced. To understand the data types, let us start with knowing Objects and its significance in python.

Objects are the basic abstraction of data in python. Data in a python program is represented as object or relations between object.

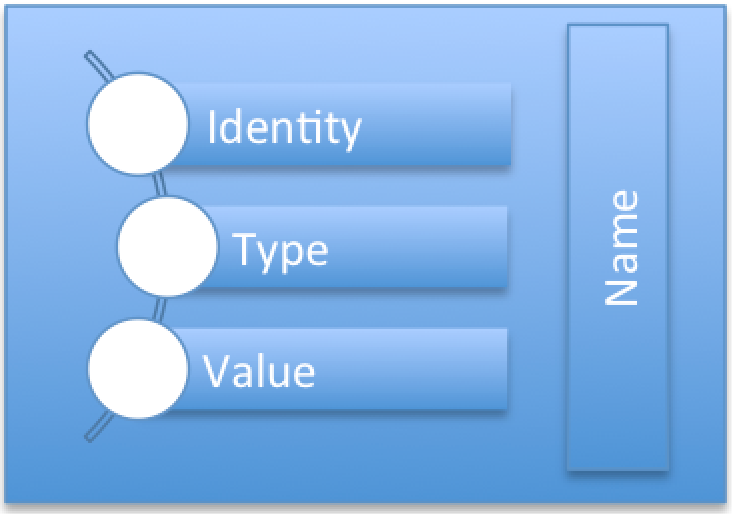
First class objects

Almost all objects in Python are first class.

Definition: An object is first class if: (1) we can put it in a structured object; (2) we can pass it to a function; and (3) we can return it from a function.

Every object has an identity, type and value associated with it.

- Identity is the identifier of the object. It can be referred as the address in memory. Identity of the object never changes during its life time.
- Object type refer to the kind of data that can be stored in it.
- Value is the content of the object.



```
number_var = 10
```

type of the object can be obtained by the function - type(obj_name)

```
>>> type(number_var)
<type 'int'>
```

Identity of the pbject can be obtained by the function - id(obj_name)

```
>>> id(number_var)
140296238158800
```

value in the variable can be accessed by its name

```
>>> number_var
10
```

Each object is referred by a name.

Naming of variables:

All variables in Python representing a data type should follow the convention mentioned below

- The first character must be an alphabet(upper or lower case) or _
- the rest can contain letters, _, or digits
- Python keywords are not allowed as name of the variable

More on variables:

- we can have our own data types
- Declaration of variables is not required in Python
- The equal "=" sign in the assignment shouldn't be seen as "is equal to". It should be "read" or interpreted as "is set to"
- Not only the value of a variable may change during program execution but the type as well. You can assign an integer value to a variable, use it as an integer for a while and then assign a string to the variable.

Naming conventions:

Python prefers the underscore method but guides developers to defer to camelCasing if integrating into existing Python code that already uses that style

```
temperature_during_morning = 24
temperatureDuringMorning = 24
```

Every data type has a characteric assigned to it and its largely governed by their ability of being modified. They are either mutable or immutable.

Mutable Vs Immutable objects:

Python has mutable and immutable data types. Objects whose value can change are said to be mutable type and objects whose value is unchangeable once they are created are called immutable.

We can test whether two names have the same value using ==:

```
>>> a == b
True
```

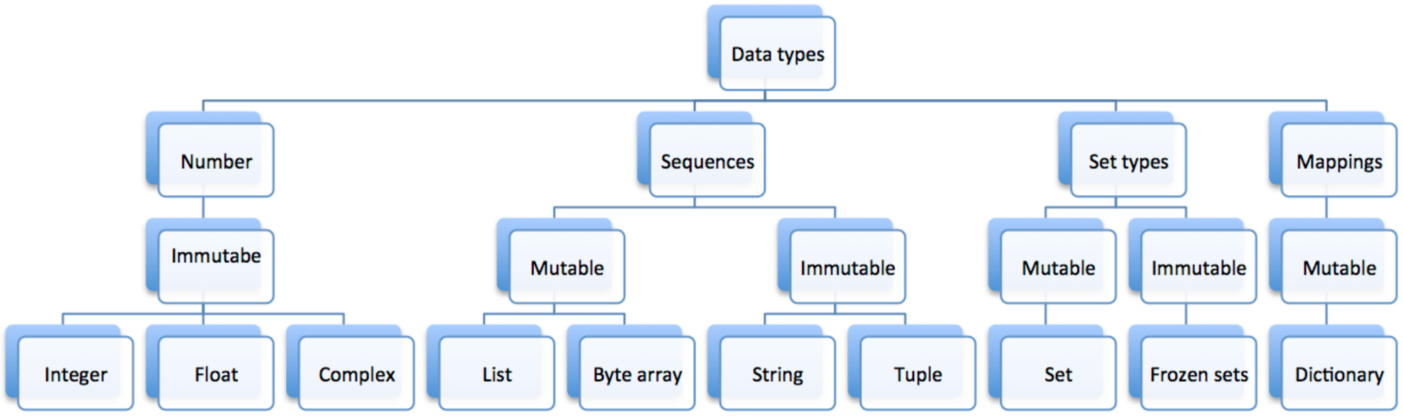
We can test whether two names refer to the same object using the is operator:

```
>>> a is b
```

False

Data Types:

Python has Numbers, Sequences, Sets and mapping categories of data types.



Numbers:

Number type objects are immutable. There are four built-in data types for numbers available in python

Integer

- They represent the normal numbers. num = 3241
- The value can be Octal literals(Base 8)

```
octal_number= 010 (Decimal 8)
```

- It can also have Hexa decimal literals *hex*anumber = 0xA01
- Its a 64 bit value (max)

Long Integers

- Numbers of unlimited size

```
long_num = 35234326677452342L
```

Floating point numbers

- float_num = 4.11 or 3.11e-10
- float_num = float(4)

Complex numbers

- (real part) + imaginary part

```
complex_numbers = 21 + 2E01
```

Sequences:

They represent finite ordered set indexed by non-negative numbers. They represent a collection of objects. We have both mutable and immutable sequences. Sequences are containers with items that are accessible by indexing or slicing.

When to use sequences?

- When you want to process a collection of items
- when you want to iterate over a collection of items
- when you want to index an item in a collection

Immutable sequences:

Strings:

Strings are represented as a contiguous set of characters. Character is a string of 1 item of 8-bit byte.

Strings can be represented with single, double and triple quotes

```
    >>> first_name = 'Tanigai'
    >>> first_name
    'Tanigai'

    >>> first_name = "Tanigai"
    >>> first_name
    'Tanigai'

    >>> first_para = ''' In the world of python programming,           object is the fundamental abstra
    >>> first_para
```

---special characters in string. ???

print "hello" 'world' ????

Unicode:

The items of an unicode object are Unicode code units.

Tuples:

Tuple is an immutable sequence. The items of a tuple are arbitrary python objects. It can collection of items separated by comma and enclosed with ().

```
>>> tup_values = (1,2,3,'Youth')
>>> type(tup_values)
<type 'tuple'>

tup_values_int = (1,2,3)
>>> type(tup_values_int)
<type 'tuple'>
```

It is possible to drop the parentheses when specifying a tuple, and only use a comma separated list of values:

```
>>> thing = 2, 4, 6, 8
>>> type(thing)
<class 'tuple'>
>>> thing
(2, 4, 6, 8)
```

Single value tuple

A single value tuple should end with a ,

```
singlue_value_tuple = (1,).
```

If it does not end with , the type of the object will be of the element in the parenthesis

```
>>> single_value_tuple = (1,)
>>> type(single_value_tuple)
<type 'tuple'>

>>> single_value = (1)
>>> type(single_value)
<type 'int'>

>>> str_value = ("you")
>>> type(str_value)
<type 'str'>
```

Empty tuple:

```
>>> a = ()
>>> type(a)
<type 'tuple'>
```

Mutable sequences:

Mutable sequences can be changed after they are created. List, Byte array are the mutable sequences available in python. Let us discuss about them in detail in the forthcoming sections

List:

It is a mutable sequence (the values in the list can be changed)

List is sqeuence that contain arbitrary python objects seperated by comma and enclosed within square bracket.

Lists are dynamic arrays - array in the sense they can be addressed using index and dynamic in the sense we can add element after it is created

Items in list need not be of the same type.

sample_list = ["USA", "India",707,60010]

- Nested lists

A nested list is a list that appears as an element in another list. In this list, the element with index 3 is a nested list:

```
>>> nested = ["hello", 2.0, 5, [10, 20]]
If we print nested[3], we get [10, 20]. To extract an element from the nested list, we can proceed in tw

>>> elem = nested[3]
>>> elem[0]
10
Or we can combine them:

>>> nested[3][1]
20
```

Bracket operators evaluate from left to right, so this expression gets the three-eth element of nested and extracts the one-eth element from it.

*Byte arrays:** ????

Created by build in bytearray() constructor.

Set types:

- They represent finite, un-ordered list of items. They are immutable.
- They cannot be indexed.
- They can be iterated.

Sets:

- It is a mutable data type.
- A set is another data structure that serves as an unordered list with no duplicate items. Below, we show how to create a set, add things to the set, test if an item is in the set, and perform common set operations (difference, intersection, union):
- Sets can't contain mutable objects, sets are mutable

```
>>> shapes = ['circle','square','triangle','circle']
>>> setOfShapes = set(shapes)
>>> setOfShapes
set(['circle','square','triangle'])

>>> setOfShapes.add('polygon')
>>> setOfShapes
set(['circle','square','triangle','polygon'])

>>> 'circle' in setOfShapes
True

>>> 'rhombus' in setOfShapes
False

>>> favoriteShapes = ['circle','triangle','hexagon']
>>> setOfFavoriteShapes = set(favoriteShapes)
>>> setOfShapes - setOfFavoriteShapes
set(['square','polyon'])

>>> setOfShapes & setOfFavoriteShapes
set(['circle','triangle'])

>>> setOfShapes | setOfFavoriteShapes
set(['circle','square','triangle','polygon','hexagon'])
```

Note that the objects in the set are unordered; you cannot assume that their traversal or print order will be the same across machines!

nums = set([1,1,2,3,3,3,4]) print len(nums) value will be 4. Set retains only unique values

Frozen sets: Immutable Frozen sets are same as sets except that they are immutable. We cannot perform upadate, delete operations on a frozen sets.

Mappings:

Dictionary

A dictionary in python is a collection of items with key value pair. Value can be any python type. Key can be any hashable python type, but usually are integer and string.

The element of a dictionary can be accessed using the key. Example dict_one['key']

It is enclosed within {}.

when to use a dictionary? - When you want to look-up by key - when you want to assign a label and use it to call an item

A dictionary is a mutable python object.

```
first_dict = {"one":1, "two":2}
>>> first_dict["one"]
1
```

With different key type

```
>>> first_dict_all = {"one":1, 2:"Two"}
>>> first_dict_all[2]
'Two'
>>> first_dict_all["one"]
1
```

With different value type

```
>>> first_dict_diff_value["one"]
1
>>> first_dict_diff_value["Two"]
' value Two'
```

Operations on Data types:

Operations on numbers:

```
 +  : addition of numbers

    a = a+b ; a+=b
 -  : subtraction of mumbers

    a = a-b; a-=b

 *  : multiplication of numbers

    a = a* b ; a*=b

 / : Division of numbers
    a = a/b ; a/= b
```

Operators on sequences:

All sequences has some common operators that can be applied on them like,

- len - len(sequene) - Returns number of items in the sequence

```
>>> len("tanigai")
7
>>> len([1,2,3])
3
>>> len((4,5,6))
3
```

- min, max operators: Returns the minimum value in the sequence

```
>>> min("tanigai")
'a'
>>> min([1,2,3])
1
>>> min((4,5,6))
4
```

Indexing a sequence:

All sequences can be accessed using index. The index starts from 0.

```
>>> name = "tanigai"
>>> name[0]
't'
```

Using an index >=L or <-L raises an exception. Assigning to an item with an invalid index also raises an exception. You can add one or more elements to a list, but to do so you assign to a slice, not an item, as I'll discuss shortly.

Concatenation & repetition

- Concatenation of 2 sequences

```
    >>> [1,2,3]+[4,5,6]
    [1, 2, 3, 4, 5, 6]

    >>> "tanigai"   +"arassane"
    'tanigaiarassane'

    >>> (3,4,5)+(5,6,7)
    (3, 4, 5, 5, 6, 7)
```

Try:

```
>>> [1,2,3]+(5,6,7)
>>> "tanigai"+[4,5,6]
```

- (n-1) repetion of the sequence , appended to the initial sequence

```
>>> "Tanigai"*2
'TanigaiTanigai'

>>> [4,5,6]*2
[4, 5, 6, 4, 5, 6]

>>> (1,2,3)*2
(1, 2, 3, 1, 2, 3)
>>>
```

Membership testing:

in, not in

```
>>> "ani" in "tanigai"
True
>>> 2 in [4,5,2]
True
>>> 3 not in (4,5,6)
True
>>>
```

Slicing a sequence:

[i,j] - supports slicing - a[i:j] returns all elements with index K such that i<= k < j

A slice is an empty subsequence if j is less than or equal to i, or if i is greater than or equal to L, the length of S.

You can omit i if it is equal to 0, so that the slice begins from the start of S. You can omit j if it is greater than or equal to L, so that the slice extends all the way to the end of S. You can even omit both indices, to mean a copy of the entire sequence: S[:].

Slicing can also use the extended syntax S[i:j:k]. k is the stride of the slice, or the distance between successive indices. S[i:j] is equivalent to S[i:j:1], S[::2] is the subsequence of S that includes all items that have an even index in S, and S[::-1] has the same items as S, but in reverse order.

- Mutable objects support deletion and immutable objects does not.
- In sequences, list supports del operation and & tuple does not.

**Specific operations: **

List: append, extend, insert, remove, pop, reverse, sort

```
L.count(x) - Returns the number of items of L that are equal to x.
L.index(x) - Returns the index of the first occurrence of an item in L that is equal to x, or raises an
L.append(x) - Appends an element to the end of list
L.extend(s) - Appends all the items of iterable s to the end
L.insert(i, x) - Inserts item x in L before the item at index i, moving following items of L (if any) “r
L.remove(x) - Removes from L the first occurrence of an item in L that is equal to x, or raises an excep
L.pop([i]) - Returns the value of the item at index i and removes it from L; if i is omitted, removes an
L.reverse( ) -  Reverses, in place, the items of L.
L.sort(cmp=cmp, key=None, reverse=False) - sorts a list
```

String: Following are some common operations in strings

```
s.lower() - -- returns the lowercase or lowercase version of the string
s.upper() -- returns the lowercase or uppercase version of the string
s.strip() -- returns a string with whitespace removed from the start and end
s.isalpha()/s.isdigit()/s.isspace( -- tests if all the string chars are in the various character classes
s.startswith('other'), s.endswith('other') -- tests if the string starts or ends with the given other st

s.find('other') -- searches for the given other string (not a regular expression) within s, and returns

s.replace('old', 'new') -- returns a string where all occurrences of 'old' have been replaced by 'new'

s.split('delim') -- returns a list of substrings separated by the given delimiter. The delimiter is not

s.join(list) -- opposite of split(), joins the elements in the given list together using the string as t
```

** % operator **

text = "%d little pigs come out or I'll %s and %s and %s" % (3, 'huff', 'puff', 'blow down')

# add parens to make the long-line work: text = ("%d little pigs come out or I'll %s and %s and %s" % (3, 'huff', 'puff', 'blow down'))

**Dictionary: **

Following are some common operations in Dictionary

```
cmp(dict1, dict2)  - Compares elements of both dict.

str(dict) - Produces a printable string representation of a dictionary

dict.clear() - Removes all elements of dictionary dict

dict.copy() - Returns a shallow copy of dictionary dict

dict.fromkeys() -Create a new dictionary with keys from seq and values set to value.

dict.get(key, default=None) - For key key, returns value or default if key not in dictionary

dict.has_key(key) - Returns true if key in dictionary dict, false otherwise

dict.items() -  Returns a list of dict's (key, value) tuple pairs

dict.keys() - Returns list of dictionary dict's keys

dict.setdefault(key, default=None) -Similar to get(), but will set

dict[key]=default if key is not already in dict

dict.update(dict2) - Adds dictionary dict2's key-values pairs to dict

dict.values() - Returns list of dictionary dict's values
```

Logical operators:

!= > < and or is in, not in = these are applied on iterable objects(sequences)

A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

**Short-circuit evaluation: ** Boolean expressions in Python use short-circuit evaluation, which means only the first argument of an and or or expression is evaluated when its value is suffient to determine the value of the entire expression.

This can be quite useful in preventing runtime errors. Imagine you want check if the fifth number in a tuple of integers named numbers is even.

The following expression will work:

```
>>> numbers = (5, 11, 13, 24)
>>> numbers[4] % 2 == 0
```

unless of course there are not 5 elements in numbers, in which case you will get:

Traceback (most recent call last): File "", line 1, in IndexError: tuple index out of range

```
>>>
```

Short-circuit evaluation makes it possible to avoid this problem.

```
>>> len(numbers) >= 5 and numbers[4] % 2 == 0
False
```

Since the left hand side of this and expression is false, Python does not need to evaluate the right hand side to determine that the whole expression is false. Since it uses short-circuit evaluation, it does not, and the runtime error is avoided.

Boolean type: Boolean refers to either Ture or False. Application f relational operattors result in boolean value.

a = True b = False

type(a) Out[10]: bool

Interesting python capabilities

Assigning multiple names to the same value

```
>>> a = b = c = 1
>>> id(a)
140599156074712
>>> id(b)
140599156074712
>>> id(c)
140599156074712
```

Assigning multiple names different values

```
>>> a,b,c = 1,2,"Youth"
>>> id(a)
140599156074712
>>> id(b)
140599156074688
>>> id(c)
4392967216
```

Swapping two numbers without using temp variable:

```
a=5


In [5]: b =2


In [6]: a,b
Out[6]: (5, 2)


In [7]: a,b=b,a


In [8]: b,a
Out[8]: (5, 2)
```

Strings and lists

Python has several tools which combine lists of strings into strings and separate strings into lists of strings.

The list command takes a sequence type as an argument and creates a list out of its elements. When applied to a string, you get a list of characters.

```
>>> list("Crunchy Frog")
['C', 'r', 'u', 'n', 'c', 'h', 'y', ' ', 'F', 'r', 'o', 'g']
```

The split method invoked on a string and separates the string into a list of strings, breaking it apart whenever a substring called the delimiter occurs. The default delimiter is whitespace, which includes spaces, tabs, and newlines.

```
>>> "Crunchy frog covered in dark, bittersweet chocolate".split()
['Crunchy', 'frog', 'covered', 'in', 'dark,', 'bittersweet', 'chocolate']
```

Here we have 'o' as the delimiter.

```
>>> "Crunchy frog covered in dark, bittersweet chocolate".split('o')
```

['Crunchy fr', 'g c', 'vered in dark, bittersweet ch', 'c', 'late'] Notice that the delimiter doesn’t appear in the list.

The join method does approximately the opposite of the split method. It takes a list of strings as an argument and returns a string of all the list elements joined together.

```
>>> ' '. join(['crunchy', 'raw', 'unboned', 'real', 'dead', 'frog'])
'crunchy raw unboned real dead frog'
```

The string value on which the join method is invoked acts as a separator that gets placed between each element in the list in the returned string.

```
>>> '**'.join(['crunchy', 'raw', 'unboned', 'real', 'dead', 'frog'])
'crunchy**raw**unboned**real**dead**frog'
The separator can also be the empty string.
```

```
>>> ''.join(['crunchy', 'raw', 'unboned', 'real', 'dead', 'frog'])
'crunchyrawunbonedrealdeadfrog'
```

he language has map(), reduce() and filter() functions; list comprehensions, dictionaries, and sets; and generator expressions.[42] The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.[43]

Exercise:

1. Syntax and operations What is the result of each of the following?

```
>>> 'Python'[1]
>>> "Strings are sequences of characters."[5]
>>> len("wonderful")
>>> 'Mystery'[:4]
>>> 'p' in 'Pineapple'
>>> 'apple' in 'Pineapple'
>>> 'pear' not in 'Pineapple'
>>> (2, 4, 6, 8, 10)[1:4]
>>> [("cheese", "queso"), ("red", "rojo")][1][0][2]
```

2. Methods and operators

```
>>> 'Python'.upper()
>>> 'Mississippi'.count('s')
>>> letters = ['c', 'z', 'b', 's']
>>> letters.sort()
>>> letters
>>> (3, 9, 8, 42, 17).index(42)
>>> "\t   \n     Don't pad me!   \n   \n".strip()
>>> mystery = 'apples pears bananas cheesedoodles'.split()
>>> mystery
>>> mystery.sort()
>>> mystery
>>> mystery.reverse()
>>> mystery
```

3. What value will appear after these two lines are entered at the prompt?

```
>>> word = "Pneumonoultramicroscopicsilicovolcanoconiosis"
>>> (word[6] + word[30] + word[33] + word[15]).upper()
```

4. Logical opposites Give the logical opposites of these conditions

```
a > b
a >= b
a >= 18  and  day == 3
a >= 18  and  day != 3
3 == 3
3 != 3
3 >= 4
not (3 < 4)
```

Strings, lists, and tuples doctest exercises

5. Four friends and a movie Write a program in a file named movie_friends.py that will produce a session something like this:

   $ python movie_friends.py

   Hmmm... You have 5 tickets to that new movie everyone wants to see. Whom should you invite to go with you?

   Enter the name of friend one: Sean Your invite list now contains: ['Sean']

   Enter the name of friend two: Jonathan Your invite list now contains: ['Sean', 'Jonathan']

   Enter the name of friend three: Margot Your invite list now contains: ['Sean', 'Jonathan', 'Margot']

   Enter the name friend four: Justin Your invite list now contains: ['Sean', 'Jonathan', 'Margot', 'Justin']

   Great! You are ready to go to the movie...

6. What will be the output of the following program?

```
this = ['I', 'am', 'not', 'a', 'crook']
that = ['I', 'am', 'not', 'a', 'crook']
print("Test 1:", this is that)
that = this
print("Test 2:", this is that)
```

References:

http://www.python-course.eu/variables.php https://developers.google.com/edu/python/introduction#indentation https://www.python.org/dev/peps/pep-0008/#introduction https://www.python.org/dev/peps/pep-0257/

https://www.safaribooksonline.com/library/view/python-in-a/0596100469/ch04s06.html http://www.diveintopython.net/native*data*types/mapping_lists.html

http://www.w3resource.com/python/python-data-type.php

Python quiz: http://www.mypythonquiz.com/question.php?qid=180

Python basics: http://ai.berkeley.edu/tutorial.html#PythonBasics

http://www.openbookproject.net/books/bpp4awd/ch03.html