Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language. Thus, **Man** and **man** are two different identifiers in Python.

Naming Conventions

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Example

myvar = “computer"

my_var = “computer"

_my_var = “computer"

myVar = “computer"

MYVAR = “computer"

myvar2 = “computer"

Keywords

- Keywords are the reserved words in Python.
- We cannot use a keyword as variable name, function name or any other identifier.
- In Python, keywords are case sensitive.
- There are 33 keywords in Python 3.7.
- This number can vary slightly over the course of time.
- All the keywords except `True`, `False` and `None` are in lowercase and they must be written as they are.

Reserved Words

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation

```
if True:
        print('Hello')
        a = 5
  b=10
```

Ex:3

```
if True: print('Hello'); a = 5 ;b=10
```

Multi-Line Statements

- Statements in Python typically end with a new line.
- Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

```
total = item_one + \
        item_two + \
        item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

Multiple statements

We can also put multiple statements in a single line using semicolons, as follows:

Example:

```
a = 1; b = 2; c = 3
```

Comments in Python

A hash sign (#) is a comment statement.

Example:

```
# First comment
print "Hello, Python!" # second comment
```

Following triple-quoted string is used as a multiline comments:

Example:

```
''' This is a novfnonfkngknfkl;
multiline comment.fm glnflkgnkf
Thank youfngklnlkflkdf
'''
```

Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

```
A=’10’
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is      made up of
multiple lines and       sentences."""
```

Python has two functions designed for accepting data directly from the user:

Syntax:

```
input(a)

raw_input("\n\nPress the enter key to exit.")
```

Variables

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Example:

```
x = 5
y = "John"
print(x)
print(y)
```

```
x = 4                # x is of type int
x = "Salary"         # x is now of type str
print(x)
```

```
x = "John“           # is the same as
x = 'John'
```

Multiple Values in single line

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

Example:

```
x = "awesome"
print("Python is " + x)
```

Let us print 8 blank lines. You can type:

```
print (8 * "\n")
```

or

```
print ("\n\n\n\n\n\n\n\n")
```

Example

```
print ("Welcome to python")
print (8 * "\n")
print (“happy morning")
```

Output:

Welcome to python

happy morning

DATA TYPES

Python has six standard data types

- Numbers
- String
- List
- Tuple
- Dictionary
- Sets
- boolean

Python Numbers

- Number data types store numeric values.

Example:

```
Sum1=10
Sum2=26
```

You can also delete the reference to a number object by using the del statement.
The syntax of the del statement is –

```
del var1[,var2[,var3[....,varN]]]]
```

You can delete a single object or multiple objects by using the del statement.
Example

```
del sum1
del sum1,sum2
```

Example

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAE l	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Example

```
str = 'Hello World!'
print str # Prints complete string
print str[0] # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
print str[2:] # Prints string starting from 3rd character
print str * 2 # Prints string two times
print str + "TEST" # Prints concatenated string
```

Output:

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python List

- List is an ordered sequence of items.
- It is one of the most used datatype in Python and is very flexible.
- All the items in a list do not need to be of the same type.
- Items separated by commas are enclosed within brackets [ ].

Example:

```
a = [1, 2.2,
'python']
```

- The index starts from 0 in Python.
- To some extent, lists are similar to arrays in C.
- One difference between them is that all the items belonging to a list can be of different data type.

- list = ['abcd', 786 , 2.23, 'john', 70.2]
- tinylist = [123, 'john']
- print list
- print list[0]
- print list[1:3]
- print list[2:]
- print tinylist * 2
- print list + tinylist

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Lists are mutable, meaning, the value of elements of a list can be altered.

```
a = [1, 2, 3]
a[2] = 4
print(a)
```

Output

```
[1, 2, 4]
```

List Length

To determine how many items a list has, use the len() function:

Example
Print the number of items in the list:

```
list1 = ["apple", "banana", "cherry"]
print(len(list1))
```

Output

3

Add Items in a list

- To add an item to the end of the list, use the append() method:

Example:

```
list1 = ["apple", "banana", "cherry"]
list1.append("orange")
print(list1)
```

Output

['apple', 'banana', 'cherry', 'orange']

add an item in a list

- To add an item at the specified index, use the insert() method:
- Example
- Insert an item as the second position:

```
list2 = ["apple", "banana", "cherry"]
list2.insert(1, "orange")
print(list2)
```

Output

['apple', 'orange', 'banana', 'cherry']

Remove an item in a list

The remove() method removes the specified item:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Output

['apple', 'cherry']

Python Tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets **([ ])** and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.
- Tuple is **immutable**

Example

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print tuple # Prints the complete tuple
print tuple[0] # Prints first element of the tuple
print tuple[1:3] # Prints elements of the tuple starting from 2nd till 3rd
```

Output

```
('abcd', 786, 2.23, 'john', 70.2)
 (abcd)
(786, 2.23)
```

Example

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print tuple[2:]
print tinytuple * 2
print tuple + tinytuple
```

Output

(2.23, 'john', 70.2)
(123, 'john', 123, 'john') ('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

Dictionary

- A dictionary is a collection which is **unordered, changeable and indexed**. In Python dictionaries are written with curly brackets, and they have keys and values.
- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.
- **Keys are unique within a dictionary while values may not be.** The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Example

```
dict = {‘class': ‘BCA', ‘year': 2, ‘subject':‘PYTHON'}
print "dict[‘class']: ", dict[‘class']
print "dict[‘year']: ", dict[‘year']
```

Output

```
dict[‘class']:BCA
dict[‘year']:2
```

Example

```
dict = {‘class':‘BCA',‘year':2,‘subject':‘PYTHON'}
dict[‘year'] = 3; # update existing entry
dict[‘college'] = “SAASC"; # Add new entry
print "dict[‘year']: ", dict[‘year']
print "dict[‘college']: ", dict[‘college']
```

Output

```
dict[‘year']: 3
dict[‘college']:  SAASC
```

Delete Dictionary Elements

- You can either remove individual dictionary elements or clear the entire contents of a dictionary.
- You can also delete entire dictionary in a single operation.

```
dict ={‘class':‘BCA',‘year':2,‘subject':‘PYTHON'}
del dict[‘class']; # remove entry with key ‘class' dict.clear(); # remove all entries in dict
del dict ; # delete entire dictionary
```

Properties:

More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

Example:

```
dict1= {'Name':‘java',‘edition':2,'Name':‘python'}
print "dict1['Name']: ", dict['Name']
```

Output

dict1['Name']:python

Sets

Set is an unordered collection of unique items.
Set is defined by values separated by comma inside braces { }.
Items in a set are not ordered.
Example:

```
a = {5,2,3,1,4} # printing set variable
print("a = ", a)
print(type(a)) # printing Datatype
```

Output

```
a = {1, 2, 3, 4, 5}
<class 'set'>
```

- We can perform set operations like union, intersection on two sets.

  Sets have unique values.

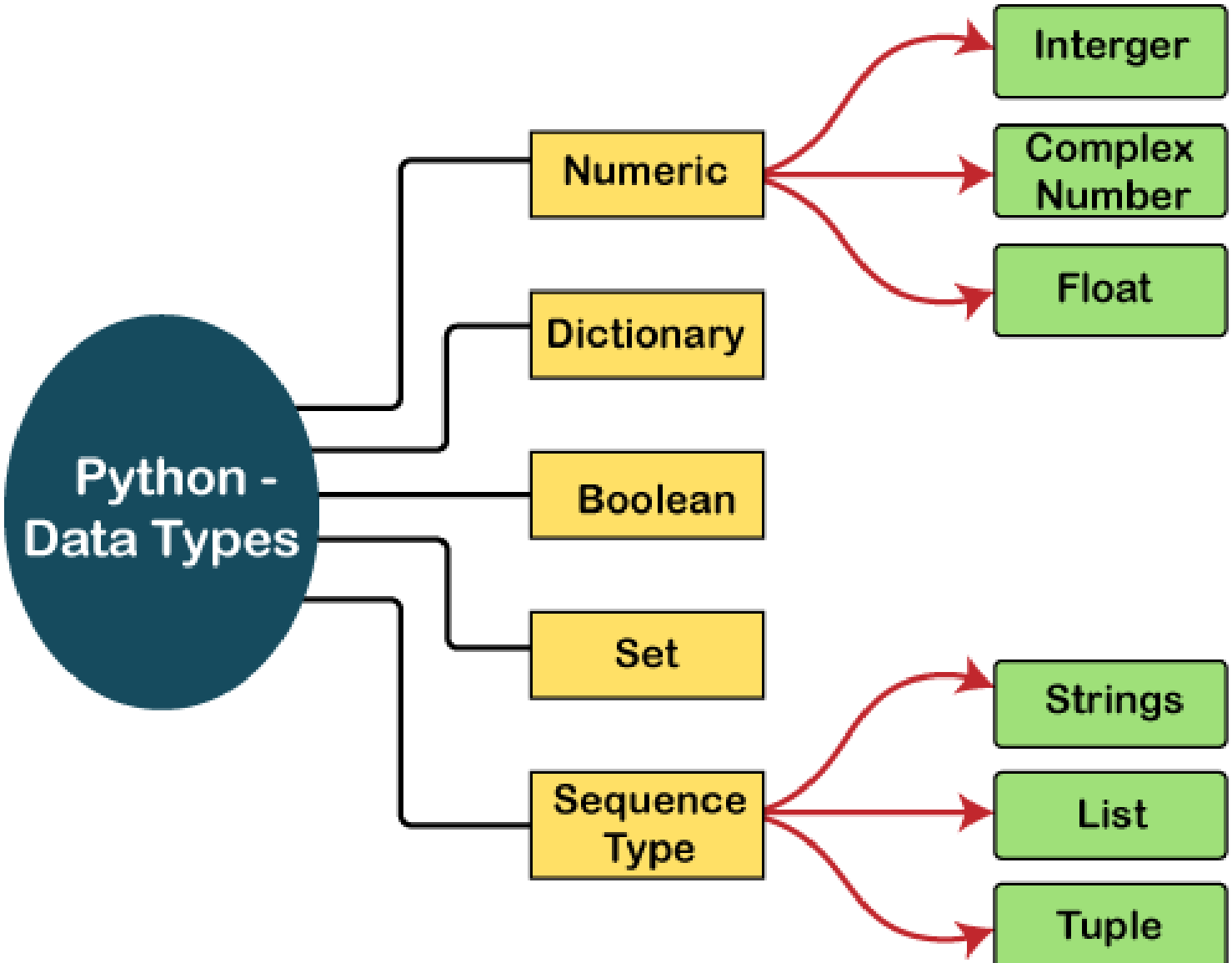
- They eliminate duplicates.

Example

```
# Creating Empty set
set1 = set()
set2 = {‘java', 2, 3,'Python'}

#Printing Set value
print(set2)

# Adding element to the set

set2.add(10)
print(set2)

#Removing element from the set
set2.remove(2)
print(set2)
```

Output

{3, 'Python', 'Java', 2} {'Python', 'Java', 3, 2, 10} {'Python', 'Java', 3, 10}


Python - Data Types
Numeric
Interger
Complex Number
Float
Dictionary
Boolean
Set
Sequence Type
Strings
List
Tuple

BOOLEAN

- Boolean type provides two built-in values, True and False.
- These values are used to determine the given statement true or false. It denotes by the class bool.
- True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'.

Example

```
print(type(True))
print(type(False))
print(false)
```

Output

```
<class 'bool'>
<class 'bool'>
NameError: name 'false' is not defined
```