Smarter education for smarter planet
IBM Innovation Centre for Education

Welcome to:

Data types & Regular expression

Unit objectives

After completing this unit, you should be able to:

- Learn all the different supported basic data types
- Understand the specialized data types script
- Learn how to traverse lists and strings
- Identify the difference between a tuple and a list
- Understand a dictionary

Simple Python Program

A Code Sample (in IDLE)

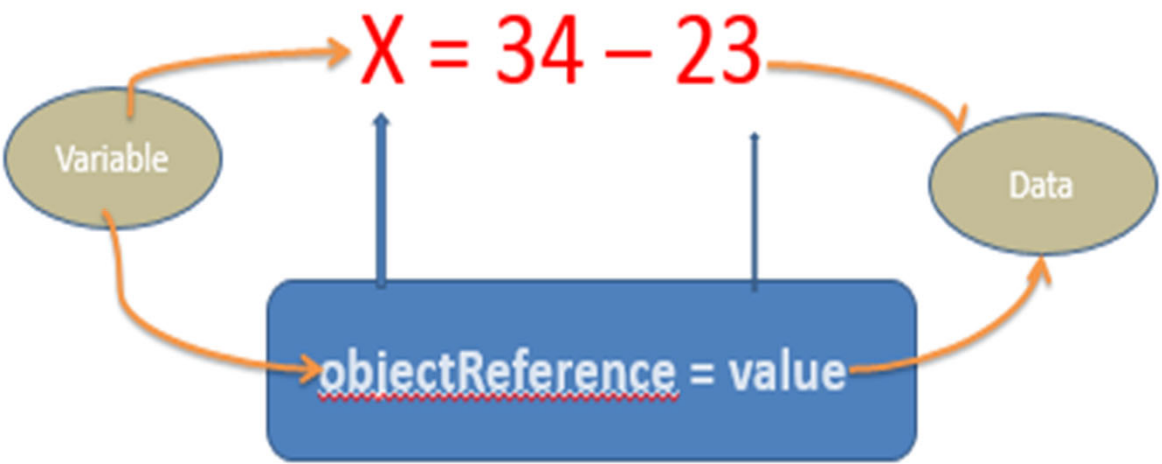
```
x = 34 - 23              # A comment.
y = "Hello"              # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"     # String concat.
print x
print y
```

Enough to Understand the Code

- Indentation matters to meaning the code
  - Block structure indicated by indentation
- The first assignment to a variable creates it
  - Dynamic typing: No declarations, names don't have types, objects do
- Assignment uses `=` and comparison uses `==`
- For numbers `+ - * / %` are as expected.
  - Use of `+` for string concatenation.
  - Use of `%` for string formatting (like printf in C)
- Logical operators are words (`and, or, not`) *not* symbols
- The basic printing command is `print`

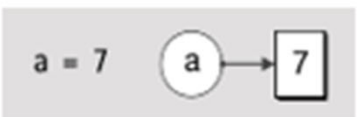
Basic Data Types Statement

The first python statement...

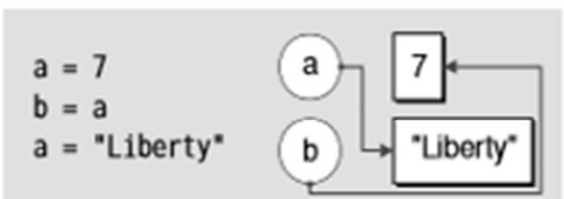


Basic Data types identifiers

Object references and objects

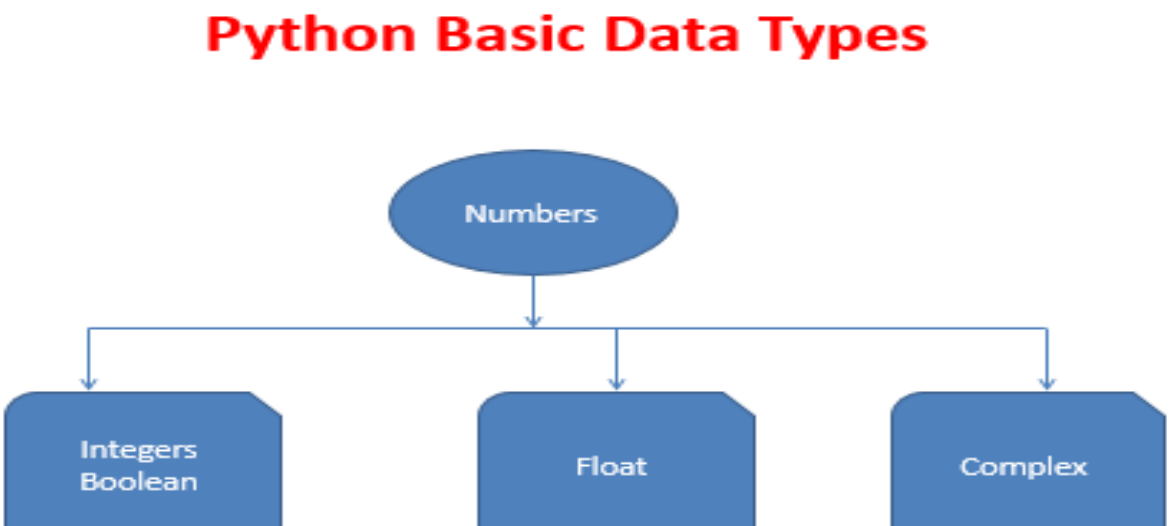


The circles represent object references.
The rectangles represent objects in memory.



Note : The name of Object reference is called as Identifiers

Basic Data types



Basic Data types Contd

- **Python** represents all its data as objects
- **Mutable objects: C**an change their content without changing their identity
  Examples : Lists & Dictionaries
- **Non-Mutable objects: C**an change their content without changing their identity

  Example : Integers, floats, strings,tuples

Integer Data Type

Integer data type

- Numbers are similar to data types of the C Programming Language
- Integer data type - does not contain any precision or decimal points.
- Boolean is a subset of Integer

Type	Format	Description
int	a = 10	Signed Integer
long	a = 345L	(L) Long integers, they can also be represented in octal and hexadecimal
float	a = 45.67	(.) Floating point real values
complex	a = 3.14J	(J) Contains integer in the range 0 to 255.

Float & Complex data type

Float

- It has a precision or a floating point
- Float internal representation depends on the platform
- Float value can be obtained from sys.float_value()

Complex

- data type has both a real and an imaginary part
- e.g: 2 + 3j , or 2 + 3J
- Note : It has to be 'j' or "J" in capital letters, and cannot be any other character.

```
>>>z = 3 + 5J
>>>print (z.real)
3.0
>>>print (z.imag)
5.0
>>>print (type ( z))
< class 'complex'>
```

Mathematical Function

- math.floor(x) - floors the value of x to the lower integer
  e.g: math.floor(1.99999) is 1.0, and is converted to float.
  Whereas int(1.99) is 1 (Integer)

- math.ceil(x) - raises the value to the nearest ceiling integer.
  e.g: math.ceil (1.0001) = 2.0

- round(x, [n]) - is another useful function to round the number to a certain precision.
  e.g: round (1.9991, 3) = 1.999
  and round (1.9999, 3) = 2.0

String Data Types

- This data type can manipulate string of characters
- Internally represented by the str() class.
- derived from the base str() object
- Written with single, double quotes, or three quotes
- All strings are stored in unicode representation internally

String Data Type Contd

- Explicitly they are prefixed with a ‘u’
- Str and unicode are both subclasses of basestring() base class
- Unicode string can be prefixed with a prefix ‘u’ or unicode(“string”)

String Manipulation Function

Some of the popular functions are

- isupper()
- islower()
- isalnum()
- isdigit()
- isspace()
- endswith(‘character’)

Sample String Python Code


```
>>>str = “python”
>>>newstr = str[3:5]
>>>print ( newstr )
h
>>>print (len(str))
6
>>>print( str.isupper() )
False
>>>print( str.islower() )
True
>>>print( endswith(‘n’) )
True
```

str[3:5]

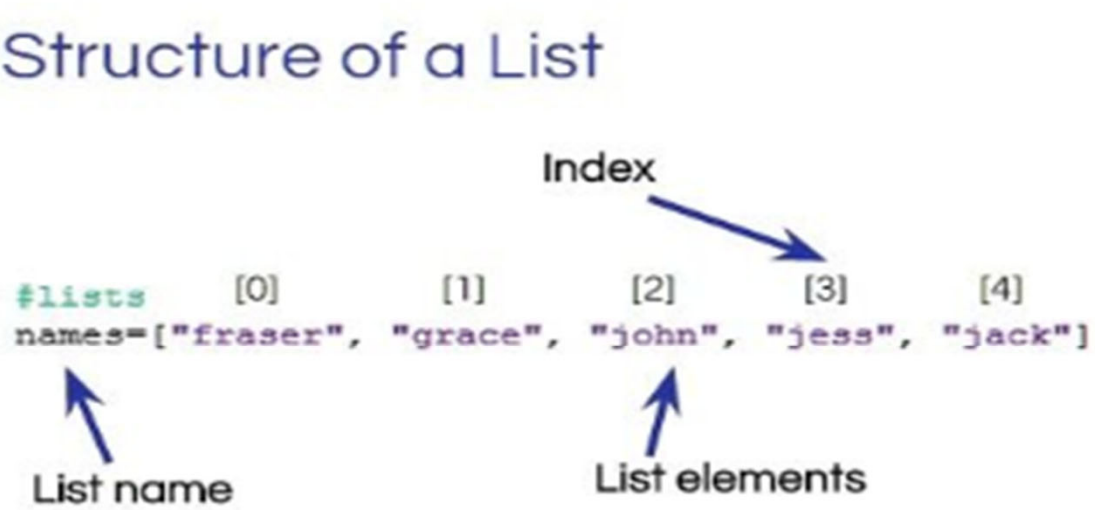
Here 3 is starting & 5 is ending range

Data types List

- A list is an ordered sequence of zero or more object references of same or different data types.
- Lists support String the same slicing and striding syntax as slicing and striding syntax strings and tuples
- Lists are enclosed in square brackets.
- List indexing starts with an index of 0, First element starts from 0

Data Types List Contd

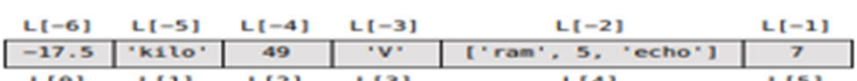
List-structure



Lists- index

- An index of -1 means - it goes like a linked list - backwards and indexes that element.
- For Example :

```
L = [-17.5, "kilo", 49, "V", ["ram", 5, "echo"], 7]
```



e.g: **L [1:3]** // prints the first, second and third.

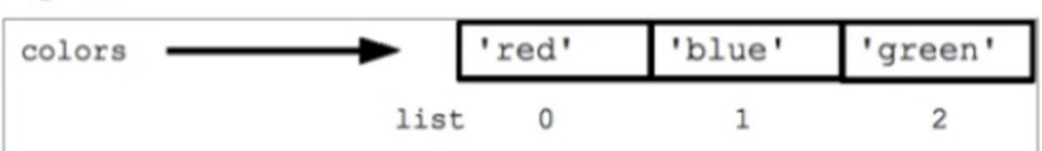
e.g: **L[1:1]** //returns an empty list

Data types List Contd

Lists

Access List Elements

```
>>> colors = ['red', 'blue', 'green']
>>> colors
['red', 'blue', 'green']
>>> colors[0]
'red'
>>> colors[1]
'blue'
>>> colors[2]
'green'
```



Data types List Contd

List - slicing

```
>>> colors = ['yellow', 'red', 'blue', 'green', 'black']
>>> colors[0:]
['yellow', 'red', 'blue', 'green', 'black']

>>> colors[:4]
['yellow', 'red', 'blue', 'green']

>>> colors[1:3]
['red', 'blue']

>>> colors[:]
['yellow', 'red', 'blue', 'green', 'black']
```

List Basic Operation

Length of a list

```
>>> colors = ['red', 'blue', 'green']
>>> len(colors)
Output:
3
```

Use `len()` function to get the length of list

List Sorting

List -sorting

- **sorted (list)** will sort the list in ascending order.
- **sorted (list, reverse=True)** will sort the list in reverse order.

Basic Sorting

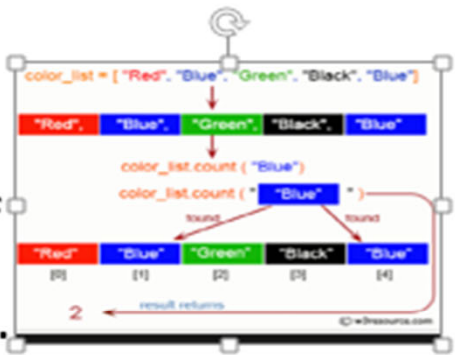
```
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> print a
[1, 2, 3, 4, 5]

>>> a = [5, 2, 3, 1, 4]
>>> a.sort(reverse=True)
>>> a
[5, 4, 3, 2, 1]

>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a.reverse()
>>> a
[5, 4, 3, 2, 1]
```

List – Count & Append

List –count & append



- **Count (param)**
  This will return the number of instances that was found that matches this param in this list.

- **Append(param)**
  Will append the element in the parameter to the end of the list

```
>>> avengers = ['Iron-man']
>>> avengers.append('Hulk')
>>> avengers
['Iron-man', 'Hulk']
>>> avengers.append('Thor')
>>> avengers
['Iron-man', 'Hulk', 'Thor']
>>>
```

Dictionary

- Dictionary is a sequence of Key:Values that are together enclosed by a {} curly braces
- Every Key has to have a value
- key has to be unique to get good results
- The Value can be ‘string’, ‘tuple’, or a number
- An empty dictionary is written as {}

```
#Dictionary Example
>>> a={"lang": "Python", "topic": "Dictionary", "category": "Data Type"}
>>> a["lang"]
'Python'
>>> a["topic"]
'Dictionary'
>>> a["category"]
'Data Type'
>>>
```

Python-Specialised data type

- Date Time
- Functions are available for this module to print the current date and time.
- User has to import date library to start using the different functions
- To get the epoch time use, time.time() function.
- The datetime module has functions and classes to parsing date and time functions
- print (dir(datetime)) will print the datetime as a method within datetime.
- print (dir(datetime.datetime)) will print the list of methods within this class.
- There is ‘now()’, prints the current local date and time. Same as ‘today()’
- The method utcnow() prints the UTC date and time

Basic Data types Collections

Collections-container data types

- This module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, dict,list,set and tuple

- Some of the common collections class are

deque( ) :list-like container with fast appends and pops on either end

deque

- A *deque* is a double-ended queue
  - Supports efficient *append* and *pop* and *appendleft* and *popleft* operations
  - Can be bounded, with overflow causing a pop from the opposite end to the append

```
>>> from collections import deque
>>> dq=deque(maxlen=3)
>>> dq
deque([], maxlen=3)
>>> dq.append(1) ; dq.append(2) ; dq.append(3)
>>> dq
deque([1, 2, 3], maxlen=3)
>>> dq.append(4)
>>> dq
deque([2, 3, 4], maxlen=3)
```

ChainMap( ) :dict-like class for creating a single view of multiple mappings

Userlist( ) :wrapper around list objects for easier list subclassing

Note : https://docs.python.org/3/library/collections.html

HEAP

- This data type can be used to create a heap, either by using a list[], or by using the **heapify()** function.

- Heaps are binary trees, Here the parent node's value is less than or equal to it's children.

  heap[k] <= heap[2*k+1] and

  heap[k] <= heap[2*k+2]

Files, Directories & Flow control

- OS.path is one of the common libraries that is used by developers

- Lets program and execute dir (os) command

```
>>>import os
>>>print ( dir(os))
['.....
_loader__', '__name__',
'__package__',
'__spec__', '_execvpe',
'_exists', '_exit',
'_fspath',
'_get_exports_list',
.......'writev']
```

Files, Directories & Flow control Contd

os.listdir

```
>>>import os
>>>print (os.listdir )
```

This will print all the contents of the current folder. (includes files and directories)

os.path.isdir (path_to_directory)

```
>>>import os
>>>print (os.path.dir ( “/var/log” )
```

Returns “True” or “False” depending if it is a directory or not

os.path.isfile(path_to_file)

```
>>>import os
>>>print (os.path.file (
“/var/log/system.log” )
```

Returns “True” or “False” depending if it is a file or not.

os.path.split (path)

```
>>>import os
>>>split_head_tail= os.path.split (
"/var/log/system.log")
>>>print (  split_head_tail[0] ) #
head i.e /var/log.
>>>print (  split_head_tail[1] )  #
tail. i.e system.log
```

os.path.getsize (path)

Return the size, in bytes, of path. Raises an OSError if the file does not exist or is inaccessible.

- stat library defines functions, constants to os.stat(), os.fstat, and os.lstat () classes.

Stat Module

```
>>>import os, sys

>>>from stat import *

>>>mode = os.stat ( "/var/log").st_mode
>>>print ( mode )
>>>if  S_ISDIR(mode):
          print " This is a directory"

>>>mode = os.stat ( "/var/log/system.log").st_mode
>>>if  S_ISREG(mode):
          print "This is a file"
```

- **stat.S_ISDIR(mode)**
  Return non-zero if the mode is from a directory.

- **stat.S_ISCHR(mode)**
  Return non-zero if the mode is from a character special device file.

- **stat.S_ISBLK(mode)**
  Return non-zero if the mode is from a block special device file.

- **stat.S_ISREG(mode)**
  Return non-zero if the mode is from a regular file.

- **stat.S_ISFIFO(mode)**
  Return non-zero if the mode is from a FIFO (named pipe).

- **stat.S_ISLNK(mode)**
  Return non-zero if the mode is from a symbolic link.

- **stat.S_ISSOCK(mode)**
  Return non-zero if the mode is from a socket.

- **stat.S_ISDOOR(mode)**
  Return non-zero if the mode is from a door.

- **stat.S_ISPORT(mode)**
  Return non-zero if the mode is from an event port.

- **stat.S_ISWHT(mode)**
  Return non-zero if the mode is from a whiteout.

Module 4

- This module helps in comparing files in directories or different directories.
- Some of the functions are :

  filecmp.cmp(f1, f2, shallow=True)

  This allows us to compare the file f1 and f2, Returns True if they are equal, Return False otherwise

- **filecmp.cmpfiles(directory1, directory2, commonname, shallow=True)**

  This function compare the files in the two directories directory1 and directory2 whose names are given by common.

- It returns three lists of file: match,mismatch, error file

Module 5

- This module finds all the files, matching a pattern as set by Unix rules.
- A typical call will look like :

```
>>>import glob
>>>files=glob.glob ( "*.py" )
>>>print ( files )
```

This takes another argument called “recursive = True”. if this is specified then the search happens for all the folders within the directory

Module 6

- This module has a number of functions to support copying and removal of files.
- In order to use the module, user would have to import the module shutil. **“ import shutil “**
- Here are some of the example function calls :

  shutil.copy(src, dst, *, follow_symlinks=True)
  This function copies the source file to the destination

- The source and the destination arguments have to be strings

Regular Expressions (RE)

- Regular Expressions are used to find certain patterns in the strings
- Used for data manipulation, text mining, using raw data to search for patterns
- Application-Machine Learning, Data processing, NLP
- Regular expressions are a small subset of the Python language to help in data manipulation

Expression Contd

- It is made available thru the “re” module
- Internally the Regular expressions is written in C Programming language and the patterns are compiled into byte codes which are executed by matching engine that is written in C.

Unicode

- In general, the Unicode versions match any character that's in the appropriate category in the Unicode database

\d ---- Matches a decimal digit, class[0-9]

\D ---- Matches any non-digit character; class [^0-9]

\s ---- Matches any whitespace character; class [\t\n\r\f\v].

\S ---- Matches any non-whitespace character; class [^ \t\n\r\f\v]

\w ---- Matches any alphanumeric character; class [a-zA-Z0-9_]

\W----Matches any non-alphanumeric character; class [^a-zA-Z0-9_]

\w+ ---Matches one or more words / characters

\b ----- boundary between word and non-word

^ ------Match the start of the string

$ ------ Match the of the string end

Note : For more details https://docs.python.org/3/howto/regex.html

Implementation of RE

- User has to include the library in the code, by specifying the **import re** command
- To process the regular expression, it has to be compiled into patterns before it can be run.e.g

```
import re;
 x = re.match ( pattern ,string )
```

Example of RE

Examples of RE

```
>>>import re
>>>p = re.match( ".*abcz", "zzabcd" )
>>>print(p)
```

Output :will print None as there is no abcz in the string "zzabcd".

Now, lets change the pattern a bit and search it in the string.
Let's make the pattern as "abcd"

```
>>>import re
>>>p = re.match( "abcz", "zzabcd" )
>>>print(p)
```

This does not match either - as the pattern 'abcd' is not in the string.The string is actually "zzabcd" and not "abcd".

now, let's match the pattern correctly

```
>>>import re
>>>p = re.match( ".*abcz", "zzabcd" )
>>>print(p)
```

The search pattern is now. .* (dot star) which means 0 or more times of the previous character that is 'dot'. A dot/period can be any character.

```
>>>import re
>>>p = re.match( ".*abcd", "zzabcd" )
>>>print(p)
```

This will return a match

Example of RE contd

Python offers two types of searching for strings.

1) match - this matches for search string from the beginning of the string.

2) the search - this searches for the search pattern in the string ANYWHERE in the string ! - this is the difference

Example 1


```
line = “Python Programming is Exciting for Engineers";

matchObj = re.match( r'Programming', line, re.M|re.I)
if matchObj:
   print "match --> matchObj.group() : ", matchObj.group()
else:
   print "No match!!"

searchObj = re.search( r'Programming', line, re.M|re.I)
if searchObj:
   print "search --> searchObj.group() : ", searchObj.group()
else:
   print "Nothing found!!”
```

Example 2


```
import re
searchpattern = re.compile ( r'.*Python', re.I|re.DEBUG )
result =  searchpattern.match( "thisisPython")

print ( result )
```

The result is :

```
MAX_REPEAT 0 MAXREPEAT
  ANY None
LITERAL 80
LITERAL 121
LITERAL 116
LITERAL 104
LITERAL 111
LITERAL 110
<_sre.SRE_Match object; span=(0, 12), match='thisisPython'>
```

Example 3


```
import re

searchpattern = re.compile ( r'\t\tPython',
re.I|re.DEBUG )
result =  searchpattern.match( "\t\tPython")

print ( result )

The result is :

LITERAL 9
LITERAL 9
LITERAL 80
LITERAL 121
LITERAL 116
LITERAL 104
LITERAL 111
LITERAL 110
<_sre.SRE_Match object; span=(0, 8),
match='\t\tPython'>
```

Example4


```
import re

searchpattern = re.compile ( '.*simple.*')
result =  searchpattern.match( 'Python is a simple language')

print ( result )
```

Results:

```
<_sre.SRE_Match object; span=(0, 27), match='Python is a
simple language'>
```

Example Regular Expression

- This function is another function in the re(gular) expression module
- The re.search() finds the first match for a pattern.
- This is different from the re.search in which it finds all the search patterns in the string.
- findall() finds *all* the matches and returns them as a list of strings, with each string representing one match.

Example 5

re.sub Regular Expression

- This substitutes the pattern with the “replacement” in the string.
- The function prototype is :

  re.sub (pattern, replacement, string, optional max =0)

- Here, the “pattern” is replaced by the “replacement” in the “string”

  print (re.sub ("hello" , "world", "hello world"))

- This prints the words “hello hello”
- This example below substitutes the Plus sign with underscore in the search pattern “555+1212”

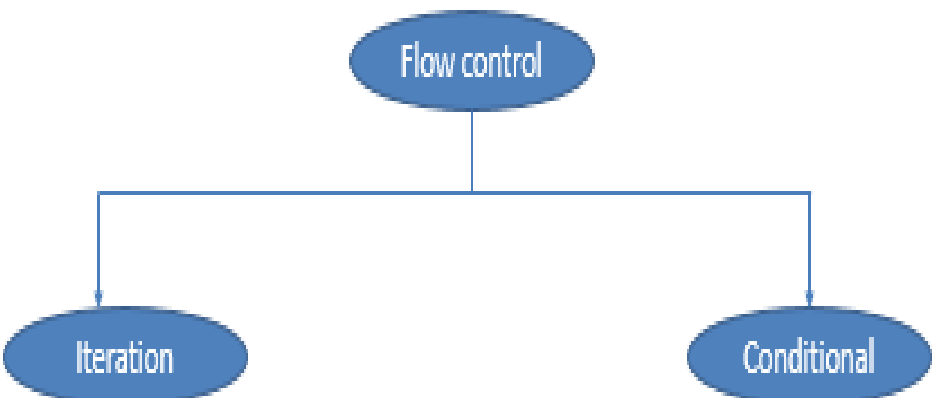
  print (re.sub ("\+" , "_", "555+1212"))

```
import re
result = re.findall(r'\w','http://www.python.org/')
print (result)

Results in :
['h', 't', 't', 'p', 'w', 'w', 'w', 'p', 'y', 't', 'h', 'o', 'n', 'o', 'r', 'g']
```

Flow Control Statements

- **Control flow statements**, however, break up the **flow** of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code



Control Statements Contd

- The flow is that the while loop has a control statement which determines how long it be in the while loop.
- break, continue, and pass statements work in conjunction with the while loop
- break is used to get out of the loop before the while statement becomes successful
- continue lets the control stay in the while loop as long as the while condition is unsuccessful.It skips the rest of the while loop.
- pass - just skips the execution

Example 1

Example 1: while loop

- An example of the while loop : This repeats until the value of x is less than 5.

```
x= 1
while ( x < 5 ) :
   x = x + 1
   print x
```

Another example of while loop: this repeats until the condition of x becomes True

```
list = ['Python', 'Programming']
x = False

while ( x== False ) :
      for item in list:
            print (item )

      x = True

print ("Completed printing the items in the
list, and out of the while loop" )
```

Result Output:

Python
Programming
Completed printing the items in the list, and out of the while loop

Example 2

Example 2 : if,else, elif condition

```
x = 4
if ( x < 10 ) :
        print ("Value of x is less than 10" )
else
        print  ("Value of x is >= than 10")
```

```
#Example with an if-elif statement

mon="Feb"
if ( mon =="Jan" ):
    print ("January")
elif ( mon == "Feb" ):
    print ("February")
elif ( mon == "Mar" ) :
    print ("March")
else :
    print ( "Other months")
```

Range function

- This is useful when stepping over a sequence of numbers
- This is used when we want to count numbers, or set a range of numbers to be printed, or processed.

This prints from 0 1o 10

```
for item in range(0,10):
    print ( item )
```

Here, the count steps by 2 every time in the loop.

```
for item in range ( 2,10, 2):
    print I
```

Output:It prints 2, 4, 6, and 8

Range Function Contd

Conditional statement

- This control statement allows for condition to be evaluated and if true, then goes to the following statement (i.e “then), if not, then it goes to “else” condition of the code.
- There is NO “then” clause in the if statement and it is assumed that the following statement after the “if” is the “then clause. (it has to be tab separated)
- The “if”, “else”, “elif” have to be ending with a colon

Checkpoint

- 1. **Find the data type of this variable.**

```
x = 100
print (type ( x))
```

- 2. **Find the data type of this variable**

```
x= 3.14
print (type ( x))
```

- 3. **Find the data type of this Boolean**

```
x = True
print ( type (x))
```

- 4. **Find the data type of this expression.**

```
x = 3+4j
print (type ( x))
```

- 5. **Find the data type of this expression**

```
x= “3 + 4j”
print (type ( x))
```

Checkpoint

6. **Find the data type of this variable**

```
x = -1
print ( type (x))
```

7. Evaluate this expression and print the result

```
x = True
print (type(x))
```

8. Divide this expression, and print the result

```
x= 3/ 2.0
print (x)
```

9. Divide the expression and print the result.

```
x = 3.0 / 2
print ( x)
```

10. Divide the expression and print the result.

```
x = 3.0 / 2
print ( x)
```

Checkpoint

11. Elements in the list can be accessed by putting the index in square brackets
    True
    False

12. Index [-1] refers to the last element in the list
    True
    False

13. Indexing in list, Index 0 means the first element
    True
    False

14. The number of elements in a list is obtained
    A.length()
    B.len(list())
    C.list.len()

15. list= [Jan", "Feb", "Mar"], list[:2] will return
    A. "Jan", "Feb"
    B. "Mar"

Checkpoint

16. For the list above, list[1:2] will return
    A. “Feb”
    B. “Feb”, “Mar”

17. list.count(“Jan”) will return
    A. 0
    B. 1

18. print (list.append(“Apr”)) will return
    A. “Apr”, “Jan” “Feb”, “Mar”
    B. “Jan”, “Feb”, “Mar”, “Apr”

19. print (list.remove(“Jan”)) will return
    A. “Feb”, “Mar”, “Apr”
    B. “Jan”, “Feb”, “Mar”, “Apr”

20. list.clear(); print list for above list will return
    A. [ ]
    B.“Jan”, “Feb”, “Mar”,

Checkpoint

2 marks questions

1. Execute this expression and print the results
(python3)

```
import math
 print ( math.floor (1.99) )
```

2. Execute this expression, and print the result

```
import math
 print round(1.999, 3)
 print round(1.9991, 3)
 print round ( 1.9999, 3)
```

3. Execute the expression and print the result.

```
 import math
  print math.ceil ( 1.9)
  print math.ceil ( 1.2)
```

Checkpoint

4. Divide this expression, and print the result

```
import math
print (3//2 )
```

5. Divide the expression and print the result.

```
import math
print ( 3 %2 )
```

6. Print the results of these multiple statements.

7. x = 1; y= 0; print (x or y)
8. x = 1; y = 1; print (x or y)
9. x = 1; y=1 ; print (x and y)
10. x = 1; y=0; print (x and y)

7. Is this a valid statement,

```
x=1; y=1; print ( x AND y )
```

Checkpoint

8. Identity the correct answer to check if a string has all Upper Case characters.

A.isUpper()

B.Upper

C.Upper()

9. Identify the correct syntax to concatenate a string in Python

A. period sign. '.' (str1.str2)

B.Plus sign. +. (str1 + str2)

C.Concat (str .concat(str2))

10. Is this a valid statement to slice a Python String, str1="Python", str2 = str1[1:4]

A.True

B.False

Checkpoint

Regular Expressions : 2 marks questions

1. Write a regular expression using re.match() to check if a pattern “programming” exists in search string.

2. Write a regular expression that ignores the case, and checks if the pattern exists in the search string.

e.g:

search string : “PYTHON is a programming language”

pattern: ‘python’

3. Write a re.match () expression to print all the debug messages as well as Ignorecase in the same expression. The search has to succeed.

Search String :

Pattern : “Python Programming”

String : “python programming is easy “

Checkpoint

Unit 2 : Regular Expressions : 4 marks questions

1.Write a python program to read a line and prints all the characters in Uppercase.

string = “python programming is easy”

2.Write a python program to print the string into lower case letters

string = ‘PYTHON PROGRAMMING IS EASY’

3. What is the difference between re.match and re.search ()

4. What is the output of group2, and group3 in this below regular expression.

```
sentence = "Python Programming is Exciting"
res     = re.match (r'(.*) (.*) (.*) (.*)', sentence )
print     ("group 1  = " +  res.group(1))
print     ("group 2  = " +  res.group(2))
print     ("group 3  = " +  res.group(3))
```

Checkpoint

5. What is the output of group3 and group4 in the below regular expression

```
sentence = "Python Programming is Exciting"
res     = re.match (r'(.*) (.*?) (.*?) (.*?)', sentence )
print     ("group 1  = " +  res.group(1))
print     ("group 2  = " +  res.group(2))
print     ("group 3  = " +  res.group(3))
print     ("group 4  = " +  res.group(4))
```

6. What is the output of the below regular expression.

```
sentence = "Python Programming is Exciting"
res     = re.match (r'(.*?)\s(.*?)\s(.*?)\s(.*)', sentence )
#print     (res.group())
print     ("group 1  = " +  res.group(1))
print     ("group 2  = " +  res.group(2))
print     ("group 3  = " +  res.group(3))
print     ("group 4  = " +  res.group(4))
```

Checkpoint

Flow Controls : 8 marks Questions

1. Write a program to calculate the H.C.F of a number ?

The H.C.F consists of two numbers (a, b) and it is the largest positive number that divides the two numbers ‘a’ and ‘b’.

2. Write a python program that will print the prime numbers from 1 to 100 in Python.

3. Write a program to print just the odd numbers in a list of number set from 1, to 100.
e.g: Input : 1, 2, 3,4, 5,…..100
Output : 1, 3,5,7 and …….99

4. Write a program to convert the temperature in Centigrade to Fahrenheit starting from 30 deg to 45 degrees C in 1 degree increments.

Unit summary

Having completed this unit, you should be able to:

- Learn all the different supported basic data types
- Understand the specialized data types script
- Learn how to traverse lists and strings
- Identify the difference between a tuple and a list
- Understand a dictionary