Introduction to Python scripting language

Sachin Deodhar

01/07/18

Content confidentiality and copyrights

- The training material and contents of this couse are for internal KPIT training purpose
- Do not share this document with anyone outside of KPIT
- No part of this document can be copied without explicit permission from KPIT

© KPIT Technologies 2018

Course structure

The course is structured is such that it contains sessions as well as practice excersises

Session 1

- Introduction and Python basics

Session 2

- Python Constructs like loops, decision making and data types like numbers and string
- Advanced objects like lists, tuples and dictonaries

Session 3

- Functions, modules, File IO and exceptions

Session 4

- Advanced topics: Classes/Objects
- Advanced topics: Networking

Session 5

- Advanced topics: Multi-threading
- Advanced topics: Reg-expressions

Excersise

- Each session will have some practice excercises which attendee can try later.

© KPIT Technologies Limited

Session 2 (1): Decision making and Loops
Session 2(2): Nums, String, List, Tuples, Dicts

Duration: 2 hours

01/07/18

Python decision making statements

Python assumes any non-zero or non-null value as TRUE while zero or null value as false. It provides following types of decision making statements.

- if statement:

This statement consiste of an expression which if evalued to TRUE, then the following statements are executed

- if else statement:

If the 'if' statement is followed by an optional else statement, then if the expression evaluates to FALSE then the statement following else are executed

- if elif else statement:

In this case elif stands for 'else if'. This is used for evaluating different expressions sequentially. This is similar to using switch/case statements.

- Nested if statements:

These are combinations of if, elif and else statement without another if/else block

 © KPIT Technologies Limited

Python decision making statements examples

Following are an examples of if, elif and else statements

```
if x < 0:
    print “x is less than zero”
elif x == 0:
    print “x is zero”
elif x == 1:
    print “x is one”
else:
    print “x is greater than one”
```

 © KPIT Technologies Limited

Python loop statements (for)

Python programming language provides following types of loops to handle iterative requirements.

for loop:

- The for loop on python is different than those in C or C++.
- In C and C++ the for loop only accepts arithmatic progressions for iterating.
- The for loop is python accepts any items in a sequence, in the order in which they appear in the sequence.
- The items could be part of a list or a string.

 © KPIT Technologies Limited

Python loop statements

It it is required to iterate through a sequence of number then Python provides a built-in function called **range()**

This function automatically generates an arithmatic progression list.

For example:

```
>>> range(5)
[0, 1, 2, 3, 4]
>>>
```

In this case a range(5) function automatically generated a list containing numbers from 0 to 4.

Using this in for loop iteration will be similar to the for loops in C or C++

 © KPIT Technologies Limited

Python loop statements (for)

Following are some example of using for loop

Using any list object

```
>>> words = ['test', 'list', 'containing', 'arbitary', 'words']
>>> for w in words:
...     print w
...
test
list
containing
arbitary
words
>>>
```

Here we can see the for loop printed one word from the list in each iteration

 © KPIT Technologies Limited

Python loop statements (for)

Following are some example of using for loop

Using range() function

```
>>> words = ['test', 'list', 'containing', 'arbitary', 'words']
>>> for indx in range(len(words)):
...     print 'Word at indx {} is {}'.format(indx, words[indx])
...
Word at indx 0 is test
Word at indx 1 is list
Word at indx 2 is containing
Word at indx 3 is arbitary
Word at indx 4 is words
```

Here we can see the for loop interated on an arithmatic index from 0 to 4. This aritmatic index is obtained using the range() funcion on the length on the list.

 © KPIT Technologies Limited

Python loop statements (while)

Python also supports traditional while loops for iterative operations. It will continue executing the statements as long as the expression provided in while evaluated to true.

Eg:

```
>>> count = 0
>>> while count < 5:
...     print count
...     count = count+1
...
0
1
2
3
4
>>>
```

 © KPIT Technologies Limited

Python break, continue and pass statements

Break statement:

In python the behaviour of the break statement is similar to that in C. It will cause the execution to break out of the innermost loop. The loop terminates and execution transfers to the statement immediately following the loop.

Continue statement:

The continue statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Pass statement:

The pass statement actually does nothing. It is used when some statement is requird syntactically but no actual action is expected.

 © KPIT Technologies Limited

Python break, continue and pass statement examples

Example of continue statement

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print "Number {} is
even".format(num)
...         continue
...     print "Number is {}".format(num)
...
Number 2 is even
Number is 3
Number 4 is even
Number is 5
Number 6 is even
Number is 7
Number 8 is even
Number is 9
>>>
```

Example of break statement

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print "Number {} is
even".format(num)
...         break
...     print "Number is {}".format(num)
...
Number 2 is even
>>>
```

Example of pass statement

```
>>> while 1:
...     pass
...

^CTraceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
>>>
```

 © KPIT Technologies Limited

Python else clauses for loops

Python loop statements may also have an else clause. It is executed when the loop terminates through exhaustion of the list (with for) or when the condition becomes false (with while). But it is not executed when the loop is terminated by a break statement. Following is an example

With else clause

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         print(n, 'is a prime number')
...
(2, 'is a prime number')
(3, 'is a prime number')
(4, 'equals', 2, '*', 2)
(5, 'is a prime number')
(6, 'equals', 2, '*', 3)
(7, 'is a prime number')
(8, 'equals', 2, '*', 4)
(9, 'equals', 3, '*', 3)
>>>
```

Without else clause

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     print(n, 'is a prime number')
...
(2, 'is a prime number')
(3, 'is a prime number')
(4, 'equals', 2, '*', 2)
(4, 'is a prime number')
(5, 'is a prime number')
(6, 'equals', 2, '*', 3)
(6, 'is a prime number')
(7, 'is a prime number')
(8, 'equals', 2, '*', 4)
(8, 'is a prime number')
(9, 'equals', 3, '*', 3)
(9, 'is a prime number')
```

 © KPIT Technologies Limited

Python number manipulation

Earlier we have seen the numeric data types. These are used to store numbers. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

Following is an example:

```
>>> myvar = 1
>>> id(myvar)
163801264
>>> myvar = 2
>>> id(myvar)
163801252
>>>
```

Where we can see, the memory address refered by myvar has changed after it was assigned a different value. (id() function returns the memory address corresponding to the variable)

The older value no longer refers to any variable and will be automaticallycleared by python garbage collection

 © KPIT Technologies Limited

Python number manipulation with built-in functions

Type conversion built-in function:

- int(x) to convert x to a plain integer.
- long(x) to convert x to a long integer.
- float(x) to convert x to a floating-point number.
- complex(x) to convert x to a complex number with real part x and imaginary part zero.
- complex(x, y) to convert x and y to a complex number with real part x and imaginary part y.

  x and y are numeric expressions

Following are two built-in numeric constants:

Pi

```
>>> from math import pi
>>> print pi
3.14159265359
>>>
```

exp

```
>>> from math import e
>>> print e
2.71828182846
>>>
```

 © KPIT Technologies Limited

Python number manipulation with built-in functions

Commonly used Mathematical built-in function:

- abs(x) - The absolute value of x
- cmp(x, y) - Returns -1 if x < y, 0 if x == y, or 1 if x > y
- max(x1, x2,...) The largest of its arguments: the value closest to positive infinity
- min(x1, x2,...) The smallest of its arguments: the value closest to negative infinity
- pow(x, y) The value of x**y.
- sqrt(x) The square root of x for x > 0
- modf(x) The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.

Other than these there are some more function like:

exp(x), ceil(x), floor(x), round(x [,n]), log(x), log10(x)

Trigonometric built-in Functions:

sin(x), cos(x), tan(x), degrees(x), radians(x), asin(x), acos(x), atan(x), etc

 © KPIT Technologies Limited

Python String manipulation

Strings are by far the most commonly used data types in Python. Simple way to assign a string to a variable is by enclosing the text in single of double quitoes. Triple quotes “”” can be used for initializing multiline verbatim strings.

Eg:

```
>>> str = "Hello"
>>> str
'Hello'
>>>

>>> str1 = 'How are you'
>>> str1
'How are you'
>>>

>>> str2 = """This is a comment
...       text, with tabs, spaces and
...       new line characters"""
>>> str2
'This is a comment\n     text, with tabs, spaces and\n     new line characters'
```

 © KPIT Technologies Limited

Python String manipulation using special operators

Following are some of the most commonly used string operators:

- % operator:

This is used mainly as a format specifier and is used to concatenate the string with the corresponding variable values. This works exactly like the format specifiers in C (printf)

Eg:

```
>>> print "My name is %s and my employee ID is %d" % ('Sachin', 123456)
My name is Sachin and my employee ID is 123456
>>>
```

Note:

In later versions of python there is a built-in function available for string formatting. It is str.format() and has a syntax {} .format(value1). The same example mentioned above using the format function is:

Eg:

```
>>> print "My name is {} and my employee ID is {}".format('Sachin', 123456)
My name is Sachin and my employee ID is 123456
>>>
```

 © KPIT Technologies Limited

Python String manipulation using special operators

Other than %, there are several other operators which are used with strings

- **Concatenation:** +, Eg: `"Hello" + "World"` will return `HelloWorld`
- **Repetation:** *, Eg: `"Hello" * 2` will return `HelloHello`
- **Slice:** [], this returns the character at the given index. Eg: str = “Hello”, then str[1] will give 'e'
- **Range slice:** [:], This will return characters from the given range. Eg: str[1:4] will give 'ell'
- **Membership: in:** This will return true if the character or string is present in the given string.

  Eg: `'l' in "Hello"` or `'ll' in "Hello"` will retrun true
- **Membership not in:** This will return true if the character of string is **not** present in the given string.

  Eg: `'q' in "Hello"` or `'abc' in "Hello"` will retrun true
- **Raw string: r/R:** This operator is used to suppress the actual meaning of escape characters

  Eg:

```
>>> print r'\n'
\n
>>>
```

 © KPIT Technologies Limited

Python String manipulation using built-in methods

Following is a list of some commonly useful built-in string functions:

isalnum( )	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
isalpha( )	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
isdigit( )	Returns true if string contains only digits and false otherwise.
isnumeric( )	Returns true if a unicode string contains only numeric characters and false otherwise.
isspace( )	Returns true if string contains only whitespace characters and false otherwise.
isupper( )	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise
islower( )	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
join(seq)	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
len( )	Returns the length of the string
lower( )/upper( )	Converts all uppercase letters in string to lowercase or all lower case to uppercase
min()/max()	Returns the min/max alphabetical character from the string str.
replace(old, new [, max])	Replaces all occurrences of old in string with new or at most max occurrences if max given.

© KPIT Technologies Limited

Python String manipulation using built-in methods

Following is a list of some more built-in string functions:

find(str, beg=0 end=len(string))	Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
count(str, beg= 0,end=len(string))	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
lstrip()/rstrip()/strip	Remove leading/trailing white spaces or strip both leading and trailing spaces
ljust/rjust(width[, fillchar])	Returns a space-padded string with the original string left/right-justified to a total of width columns.
decode(encoding='UTF-8',errors='strict')	Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
encode(encoding='UTF-8',errors='strict')	Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.

There are some more string function which are uncommon so not included in the list. Python document can be referred to get more information on those.

 © KPIT Technologies Limited

Python Lists

- The list is a most versatile datatype available in Python which can be written as a list of comma separated values (items) between square brackets.
- Important thing about a list is that items in a list need not be of the same type.
- Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Example of List

```
list1 = ['python', 'list example', 2, 20];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"]
```

Common operations on Lists

```
>>> list1 = ['python', 'list example', 2, 20];
>>> print "list1[1]: ", list1[1]
list1[1]:  list example
>>> list2 = [1, 2, 3, 4, 5, 6, 7 ];
>>> print "list2[1:5]: ", list2[1:5]
list2[1:5]:  [2, 3, 4, 5]
```

 © KPIT Technologies Limited

Python Lists

Following is an example for updating a list

```
>>> list1 = ['python', 'list example', 2, 20];
>>> print list1
['python', 'list example', 2, 20]
>>> list1[1] = 2
>>> list1[2] = 10
>>> list1[3] = "New string"
>>> print list1
['python', 2, 10, 'New string']
>>>
```

Here we can see that the individual items in the list could be updated. Also the type of the updated items was also different that the original item types.

© KPIT Technologies Limited

Python Lists

Following is an example of appending and deleting an item in the list

```
>>> list1 = ['python', 'list example', 2, 20];
>>> print list1
['python', 'list example', 2, 20]
>>> list1 = list1 + ["New string"]
>>> print list1
['python', 'list example', 2, 20, 'New string']
```

Following is an example for deleting an item from a list (considering the same list as above)

```
>>> del(list1[1])
>>> print list1
['python', 2, 20, 'New string']
>>>
```

We can see the 1st index corresponding to item 'list example' is now deleted

 © KPIT Technologies Limited

Python Lists operators

Following are basic list operators:

Concatenation (+): Concatenate two lists

Repetation (*): Repeat the same item in the list

Membership (in): Returns true if the item is in the list

Iteration: This is an internal operator used on for loops.

Indexing: The items of the list can be accessed using its positional index. Negative index would count the list items from the end.

Slicing: [:] operator can be used to slice the list, same as with strings

Following are some built-in functions which operate on the lists:

cmp(list1, list2) - Compares elements of both lists

max/min(list) - Returns item from the list with max/min value.

len(list) - Return the length of the string

© KPIT Technologies Limited

Python Lists methods

Following are the methods available with list objects:

`list.append(obj)`	This will append an object obj to list. Similar to '+'
`list.count(obj)`	Returns count of how many times obj occurs in list. Similar to len()
`list.extend(seq)`	This will appends the contents of seq to list
`list.index(obj)`	Returns the lowest index in list that obj appears
`list.insert(index, obj)`	Inserts object obj into list at offset index
`list.pop(obj=list[-1])`	Removes and returns last object or obj from list
`list.remove(obj)`	Removes object obj from list
`list.reverse()`	Reverses objects of list in place
`list.sort([func])`	Sorts objects of list, use compare func if given

© KPIT Technologies Limited

Python Tuples

A tuple is a sequence of immutable Python objects. They are similar to lists but they cannot be changed. Unlike square brackets in case of lists, the tuples are represented with parentheses.

Eg:

```
>>> tup1 = ('Hello', 'python', 2.7, 3.0);
>>> tup2 = (1, 2, 3, 4, 5 );
>>> tup3 = "a", "b", "c"
>>> print tup3
('a', 'b', 'c')
```

NOTE: A comma separated set of values can also be used to initialize a tuple. The parentheses can be dropped.

 © KPIT Technologies Limited

Python Tuples – Updating and deleting

Tuples are immutable, which means the actual values of the tuple elements cannot be change.

However portions of existing tuples can be extracted to create new tuples.

Eg:

```
>>> tup1 = ("a", "b", "c", "d", "e");
>>> tup2 = (1, 2, 3, 4, 5 );
>>> tup3 = tup1 + tup2
>>> print tup3
('a', 'b', 'c', 'd', 'e', 1, 2, 3, 4, 5)
>>>
```

Deleting an individual tuple element is not possible, however entire tuple can be deleted.

Eg: Deleting the previously created tup3

```
>>> del tup3
>>> print tup3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'tup3' is not defined
>>>
```

 © KPIT Technologies Limited

Python Tuples – Operators

The supported operators same as that of python Lists

Concatenation (+): Concatenate two lists

Repetation (*): Repeat the same item in the list

Membership (in): Returns true if the item is in the list

Iteration: This is an internal operator used on for loops.

Indexing: The items of the list can be accessed using its positional index. Negative index would count the list items from the end.

Slicing: [:] operator can be used to slice the list, same as with strings

Following are some built-in functions which operate on the lists:

cmp(list1, list2) - Compares elements of both lists

max/min(list) - Returns item from the list with max/min value.

len(list) - Return the length of the string

© KPIT Technologies Limited

Python Dictionaries

Dictionary in python is essentially a list of key value pairs. Unlike lists and tuples, dicts are enclosed in curly brackets.

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.

Eg:

```
>>> dict = {'Name': 'Sachin', 'Surname': 'Deodhar', 'Id': 123456}
>>> dict
{'Surname': 'Deodhar', 'Name': 'Sachin', 'Id': 123456}
>>> dict['Name']
'Sachin'
>>> dict['Id']
123456
>>>
```

 © KPIT Technologies Limited

Python Dictionaries – Updating the dict entry

An empty dictionary can be created using the following command

```
>>> mydict = {}
>>> print mydict
{}
>>>
```

Key value pairs can be added to this dictionary using the following commands.

```
>>> mydict['Name'] = "Sachin"
>>> mydict['Surname'] = "Deodhar"
>>> mydict['Id'] = 123456
>>> mydict
{'Surname': 'Deodhar', 'Name': 'Sachin', 'Id': 123456}
>>>
```

Similary existing entry can be modified using

```
>>> mydict['Id'] = 987654
>>> mydict
{'Surname': 'Deodhar', 'Name': 'Sachin', 'Id': 987654}

>>>
```

 © KPIT Technologies Limited

Python Dictionaries – Deleting dictionary elements

It is possible to delete individual entries from the dictionary

```
>>> mydict
{'Surname': 'Deodhar', 'Name': 'Sachin', 'Id': 987654}
>>> del mydict['Id']
>>> mydict
{'Surname': 'Deodhar', 'Name': 'Sachin'}
>>>
```

It is also possible to clear all entries from the dictionary

```
>>> mydict
{'Surname': 'Deodhar', 'Name': 'Sachin', 'Id': 987654}
>>> mydict.clear()
>>> mydict
{}
>>>
```

Entire dictionary itself can also be deleted

```
>>> del(mydict)
>>> mydict
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'mydict' is not defined
>>>
```

 © KPIT Technologies Limited

Python Dictionaries – Keys and built-in functions

- The dictonary keys need to be unique.

- When duplicate keys encountered during assignment, the last assignment wins

- Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys

Built-in dictionary functions

`cmp(dict1, dict2)`	Compares elements of both dict.
`len(dict)`	Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
`str(dict)`	Produces a printable string representation of a dictionary
`type(variable)`	Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

 © KPIT Technologies Limited

Python Dictionaries – methods

Python includes following dictionary methods

`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

© KPIT Technologies Limited

Thank you

01/07/18