Future Skills Training Program

Data Science and Analytics

Topic:- Basics of Python

Dr. Sunil Joshi

Agenda

Python Basic

- List
- Tuple
- Dictionary
- Function
- Module

Python 3 – List

- The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
- Python has six built-in types of sequences, but the most common ones are lists and tuples.

Python 3 – List

- **The list is the 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 the items in a list need not be of the same type.**

```
list1= ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

Accessing Values in Lists

- To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

```
list1 = [‘maths', ‘science', 2019, 2020]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5]);
```

Updating Lists

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.

```
list1 = [‘maths', ‘science', 2019, 2020]
print ("Value available at index 2 : ", list1[2])
list1[2] = 2021
print ("New value available at index 2 : ", list1[2])
```

Delete List Element

- To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting.
- You can use the remove() method if you do not know exactly which items to delete..

```
list = [‘maths', ‘science', 2019, 2020]
print (list)
del list[2]
print ("After deleting value at index 2 : ", list)
```

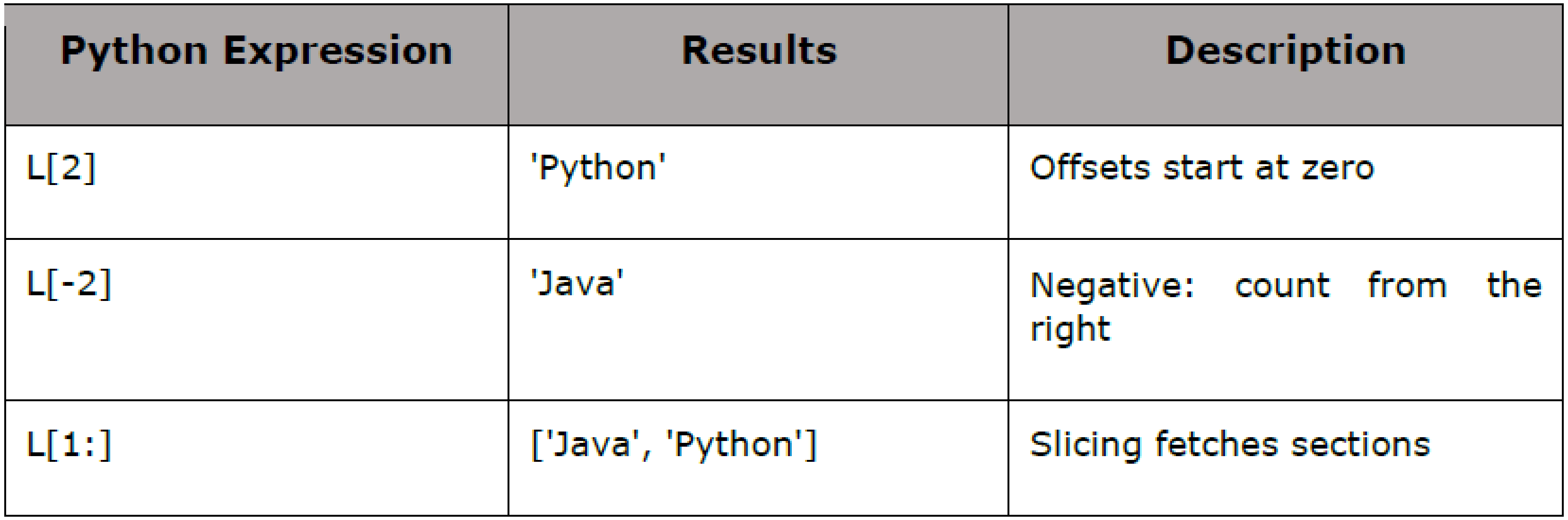
Basic List Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1,2,3] : print (x,end=' ')	1 2 3	Iteration

Indexing, Slicing and Matrixes

- **Since lists are sequences, indexing and slicing work the same way for lists as they do for strings.**

L=['C++", 'Java', 'Python'].



Python Expression	Results	Description
L[2]	'Python'	Offsets start at zero
L[-2]	'Java'	Negative: count from the right
L[1:]	['Java', 'Python']	Slicing fetches sections

Built-in List Functions & Methods

SN	Function with Description
1	**cmp(list1, list2)** No longer available in Python 3.
2	**len(list)** Gives the total length of the list.
3	**max(list)** Returns item from the list with max value.
4	**min(list)** Returns item from the list with min value.
5	**list(seq)** Converts a tuple into list.

List Methods

SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index

List Methods

6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

Python 3 – Tuples

- A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists.
- The main difference between the tuples and the lists is that the tuples cannot be changed unlike lists.
- Tuples use parentheses, whereas lists use square brackets..

```
tup1= ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"
```

Accessing Values in Tuples

- **To access values in tuple, use the square brackets for slicing along with the index or indices to obtain the value available at that index.**

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

Updating Tuples

- **Tuples are immutable, which means you cannot update or change the values of tuple elements.**
- **You are able to take portions of the existing tuples to create new tuples as the following example demonstrates.**

```
tup1 = (12, 34.56)
# Following action is not valid for tuples
 tup1[0] = 100;
```

Delete Tuple Element

- **Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.**
- **To explicitly remove an entire tuple, just use the del statement.**

```
tup = ('physics', 'chemistry', 1997, 2000);
print (tup)
del tup;
print "After deleting tup : "
print tup
```

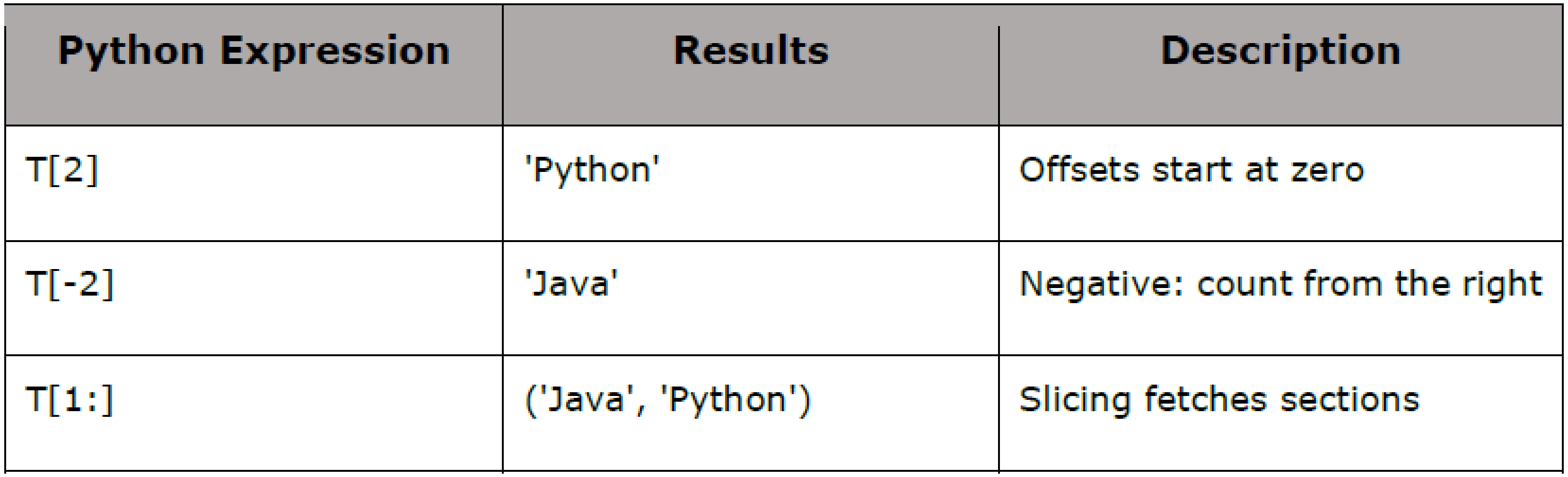
Basic Tuple Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1,2,3) : print (x, end=' ')	1 2 3	Iteration

Indexing, Slicing and Matrixes

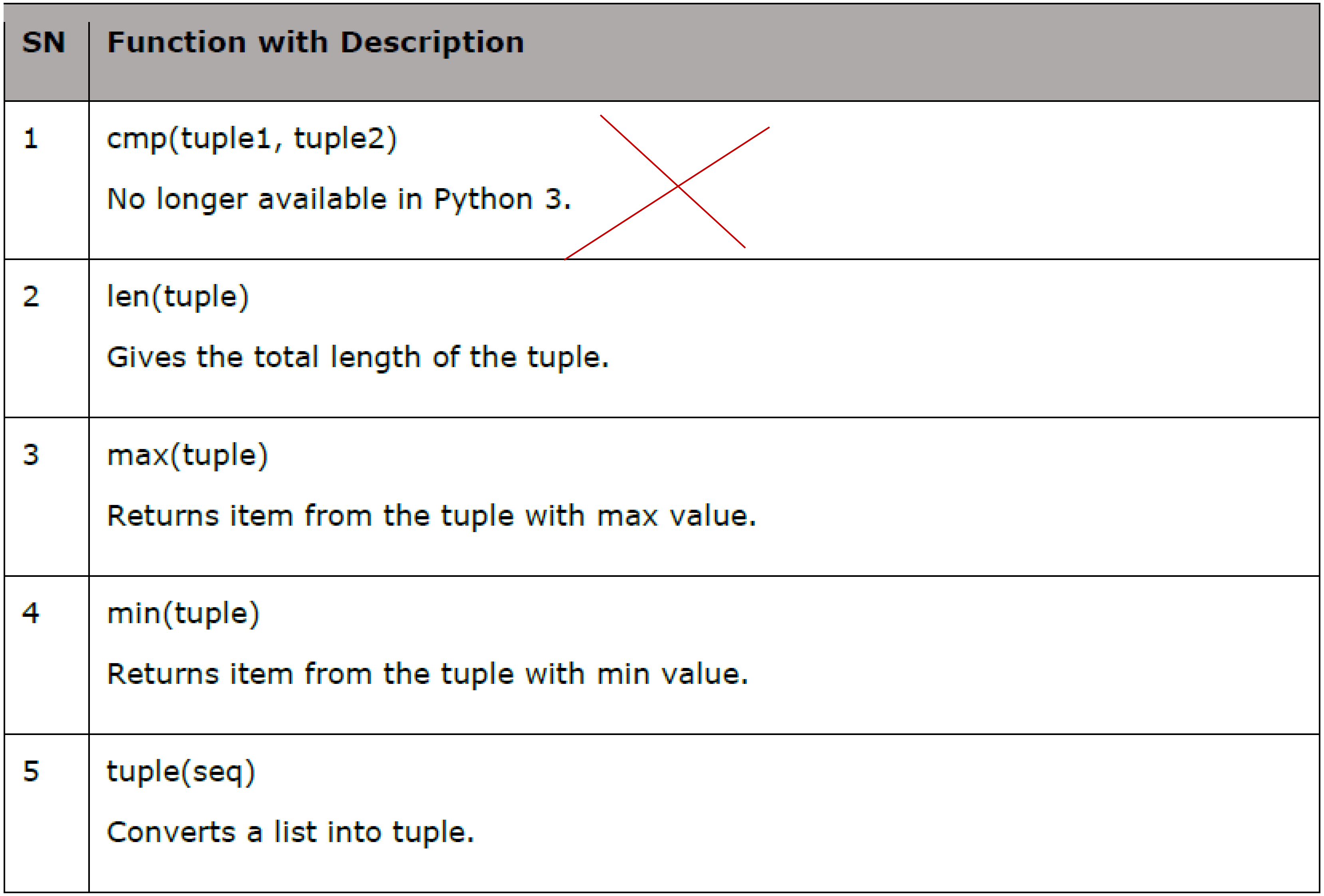
- **Since lists are sequences, indexing and slicing work the same way for lists as they do for strings.**

L=['C++", 'Java', 'Python'].



Python Expression	Results	Description
T[2]	'Python'	Offsets start at zero
T[-2]	'Java'	Negative: count from the right
T[1:]	('Java', 'Python')	Slicing fetches sections

Built-in Tuple Functions



SN	Function with Description
1	cmp(tuple1, tuple2) No longer available in Python 3.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

Python 3 – Dictionary

- **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.**

```
dict = {'Name': ‘Sunil', 'Age': 42, 'Class': 'First'}
print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])
```

Updating Dictionary

- **You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown in a simple example given below.**

```
dict = {'Name': ‘Ajay', 'Age': 27, 'Class': 'First'}
dict['Age'] = 28; # update existing entry
dict['School'] = "DPS School" # Add new entry
print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

Delete Dictionary Element

- 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.
- To explicitly remove an entire dictionary, just use the del statement.

```
 dict = {'Name': ‘Ajay', 'Age': 27, 'Class': 'First'}
del dict['Name'] # remove entry with key 'Name'
dict.clear() # remove all entries in dict
del dict # delete entire dictionary
print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

Built-in Tuple Functions

SN	Functions with Description
1	**cmp(dict1, dict2)** No longer available in Python 3.
2	**len(dict)** Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	**str(dict)** Produces a printable string representation of a dictionary.
4	**type(variable)** Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Dictionary Methods

SN	Methods with Description
1	**dict.clear()** Removes all elements of dictionary *dict.*
2	**dict.copy()** Returns a shallow copy of dictionary *dict.*
3	**dict.fromkeys()** Create a new dictionary with keys from seq and values *set* to *value*.
4	**dict.get(key, default=None)** For *key* key, returns value or default if key not in dictionary.

Dictionary Methods

5	**dict.has_key(key)** Removed, use the **in** operation instead.
6	**dict.items()** Returns a list of *dict*'s (key, value) tuple pairs.
7	**dict.keys()** Returns list of dictionary dict's keys.
8	**dict.setdefault(key, default=None)** Similar to get(), but will set dict[key]=default if *key* is not already in dict.
9	**dict.update(dict2)** Adds dictionary *dict2*'s key-values pairs to *dict.*
10	**dict.values()** Returns list of dictionary *dict*'s values.

Python 3 – Function

- **A function is a block of organized, reusable code that is used to perform a single, related action.**
- **Functions provide better modularity for your application and a high degree of code reusing.**
- **Python gives you many built-in functions like print(), etc.**
- **but you can also create your own functions. These functions are called user-defined functions.**

Rules of Defining a Function

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses.
- You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.

Syntax

```
def functionname( parameters ):
        "function_docstring"
        function_suite
        return [expression]
```

Example

```
def  display ( str ):
        “This is function to display "
        print(str)
        return
```

Calling a Function

- **Defining a function gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.**
- **Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.**
- **Following is an example to call the printme() function-**

```
# Function definition is here
def  display( str ):
        "This function prints a message"
        print (str)
        return
# Now you can call function
display ("This is first call ")
 display ("Again second call ")
```

Pass by Reference vs Value

- All parameters (arguments) in the Python language are passed by reference.
- It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

```
 # Function definition is here
def modifyme( mylist ):
        "This function modify list"
        print ("Values before change: ", mylist)
        mylist[2]=50
        print ("Values after change: ", mylist)
        return
```

Pass by Reference vs Value

```
# Now you can call modifyme function
mylist = [10,20,30]
modifyme( mylist )
print ("Values outside the function: ", mylist)
```

Function Argument

Required arguments

Keyword arguments

Operators

Default arguments

Variable-length arguments

Required arguments

- **Required arguments are the arguments passed to a function in correct positional order.**
- **Here, the number of arguments in the function call should match exactly with the function definition.**
- **To call the function printme(), you definitely need to pass one argument, otherwise it gives a syntax error as follows-**

```
# Function definition is here
def  display( str ):
        "This function prints a message"
        print (str)
        return
# Now you can call function
display ()
```

Keyword Arguments

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.
- You can also make keyword calls to the display() function in the following ways-

```
# Function definition is here
def  display( str ):
        "This function prints a message"
        print (str)
        return
# Now you can call function
display (str="Hello Students")
```

Keyword Arguments

```
# Function definition is here
def  displayinfo( name, age ):
        "This prints a passed info into this function"
        print ("Name: ", name)
        print ("Age ", age)
        return
# Now you can call printinfo function
displayinfo( age=42, name=“sunil" )
```

Default Arguments

- **A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.**
- **The following example gives an idea on default arguments, it prints default age if it is not passed.**

```
# Function definition is here
def displayinfo( name, age = 35 ):
        "This prints a passed info into this function"
        print ("Name: ", name)
        print ("Age ", age)
        return
# Now you can call printinfo function
displayinfo( age=47, name="ajay" )
displayinfo( name="rahul" )
```

Variable-length Arguments

- You may need to process a function for more arguments than you specified while defining the function.
- These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.
- Syntax for a function with non-keyword variable arguments is given below

```
def functionname([formal_args,] *var_args_tuple ):
  "function_docstring"
  function_suite
  return [expression]
```

Variable-length Arguments

- **An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments.**
- **This tuple remains empty if no additional arguments are specified during the function call.**

```
# Function definition is here
def printinfo( arg1, *vartuple ):
        "This prints a variable passed arguments"
        print ("Output is: ")
        print (arg1)
        for var in vartuple:
                print (var)
return
# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

The Anonymous Functions

- These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.
  - Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
  - An anonymous function cannot be a direct call to print because lambda requires an expression.
  - Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
  - Although it appears that lambdas are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is to stack allocation by passing function, during invocation for performance reasons.

The Anonymous Functions

lambda [arg1 [,arg2,.....argn]]:expression

Example of Anonymous Functions

```
 # Function definition is here
sum = lambda arg1, arg2: arg1 + arg2
# Now you can call sum as a function
print ("Value of total : ", sum( 10, 20 ))
print ("Value of total : ", sum( 20, 20 ))
```

The return Statement

- **The statement return [expression] exits a function, optionally passing back an expression to the caller.**
- **A return statement with no arguments is the same as return None.**
- **All the examples given above are not returning any value. You can return a value from a function as follows-**

```
# Function definition is here
def sum( arg1, arg2 ):
        # Add both the parameters and return them."
        total = arg1 + arg2
        print ("Inside the function : ", total)
        return total
# Now you can call sum function
total = sum( 10, 20 )
print ("Outside the function : ", total )
```

Scope of Variables

- **All variables in a program may not be accessible at all locations in that program.**
- **This depends on where you have declared a variable.**
- **The scope of a variable determines the portion of the program where you can access a particular identifier.**
- **There are two basic scopes of variables in Python-**
  - Global variables
  - Local variables

Global vs. Local variables

- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
- This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.
- When you call a function, the variables declared inside it are brought into scope.

Global vs. Local variables

Following is a simple example-

```
total = 0 # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
        # Add both the parameters and return them."
        total = arg1 + arg2; # Here total is local variable.
        print ("Inside the function local total : ", total)
        return total
# Now you can call sum function
sum( 10, 20 )
print ("Outside the function global total : ", total )
```

Python 3 – Modules

- **A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use.**
- **A module is a Python object with arbitrarily named attributes that you can bind and reference.**
- **Simply, a module is a file consisting of Python code. A module can define functions, classes and variables.**
- **A module can also include runnable code.**

Example

The Python code for a module named aname normally resides in a file namedaname.py.

Here is an example of a simple module, support.py

```
def  print_func( par ):
   print "Hello : ", par
   return
```

The import Statement

- You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax

```
import module1[, module2[,... moduleN]
```

- When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module hello.py, you need to put the following command at the top of the script-

```
# Import module support
import support
# Now you can call defined function
support.print_func(“Raja")
```

The from...import Statement

- **Python's from statement lets you import specific attributes from a module into the current namespace.**
- **The from...import has the following syntax**

  from modname import name1[, name2[, ... nameN]]

Example

- For example, to import the function fibonacci from the module fib, use the following statement-

```
# Fibonacci numbers module
def fib(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
            result.append(b)
            a, b = b, a+b
    return result
==============================================

    from fib import fib
    fib(100)
```

The from...import * Statement:

- It is also possible to import all the names from a module into the current namespace by using the following import statement

```
from modname import *
```

- This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

Executing Modules as Scripts

- **Within a module, the module’s name (as a string) is available as the value of the global variable __name__. The code in the module will be executed, just as if you imported it, but with the __name__ set to "__main__".**

```
from modname import *
# Fibonacci numbers module
def fib(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
if __name__ == "__main__":
    f=fib(100)
    print(f)
```

The dir( ) Function

- The dir() built-in function returns a sorted list of strings containing the names defined by a module.
- The list contains the names of all the modules, variables and functions that are defined in a module. Following is a simple example-

```
# Import built-in module math
import math
content = dir(math)
print (content)
```

Home Work 1- List

Q.1 write a program to remove duplicate from list.
Q.2 Python program to interchange first and last elements in a list.
Q.3. Python program to swap two elements in a list
Q.4. Python | Ways to find length of list
Q.5. Python | Ways to check if element exists in list
Q.6. Python | Reversing a List
Q.7. Python | Count occurrences of an element in a list
Q.8. Python program to find sum of elements in list
Q.9. Python | Multiply all numbers in the list
Q.10. Python program to find smallest number in a list
Q.11. Python program to find largest number in a list
Q.12. Python program to find second largest number in a list
Q.13. Python program to find N largest elements from a list
Q.14. Python program to print even numbers in a list
Q.15. Python program to count Even and Odd numbers in a List
Q.16. Python program to print positive numbers in a list
Q.17. Python program to count positive and negative numbers in a list

Home Work 2

Q.1 Create a list of tuples from given list having number and its cube in each tuple

Q.2 Sort a list of tuples by second Item

Q.3 Python Program to Sort Python Dictionaries by Key or Value

Q.4 Python program to find the sum of all items in a dictionary

Q.5 Python program for Merging two Dictionaries

Q.6 Python Program to Find LCM using function

Q.7 Python Program to Find HCF using function

Q.8 Python Program to Convert Decimal to Binary, Octal and Hexadecimal using function

Q.9 Python Program To Find ASCII value of a character

Q.10 Python Program to Make a Simple Calculator

Q.11 Python Program to Display Calendar

Q.12 Python Program to Display Fibonacci Sequence Using Recursion

Q.13 Python Program to Find Factorial of Number Using Recursion

?

suniljoshi.cse@satiengg.org

Thank You