OBJECT ORIENTED PROGRAMMING

Functions

Zuhaib A. Shaikh,
Asst. Prof., CSE Deptt.,QUEST

Web: zuhaib-shaikh.neocities.org

Functions

- Small set of code
- To be defined before calling it
- Can take many arguments and return many values
  - Syntax:

```
def function_name(arguments_optional):
        function_body
        return (values_optional)
```

- * can be used in the argument when programmer is unaware about number of arguments
  - e.g. def func (a, *b):
  - If more than two arguments are passed, then those will be stored as tulip in b
- * can be used before an argument if the programmer needs to mention the argument name and its value in call:
  - E.g. def func(a,*,b):
  - In the function call, second argument should be with the argument name i.e. func(1, b=2)
- A function can be assigned and can be called from a reference
  - e.g. F=fun
  - F(1,b=2)

Functions

- Default value arguments can be used
  - If argument is passed, then that value will be used else default value will be considered
  - e.g. def func(a, b=2):
- All function uses their own local variables
- If a function tries to change a global variable, then a dynamic temporary variable is created with similar reference
  - no change in global variable contents
- Global keyword can be used to create new temporary variable reference to actual global variable
- Anonymous or Inline functions can also be used
  - Called Lambda function in python
  - Lambda function can return values directly thus no need to write return statement
  - Syntax:

    function_name = **lambda** *arguments_optional* : function_body

    e.g.

    num = **lambda** *x* : “Even” if x%2==0 else “Odd”

Functions

- Map and filter function can be used to apply function on list
  - E.g.

list(map(num, [1,2,3]**)** – apply function on list and results list of returned values

list(map(lambda x: x%2, [1,2,3]**)** – apply function and result list of elements who's return value is 1

- Sub functions can be created inside a function
  - Which are then attached an as attribute for calling with function name
  - E.g.

```
def func(a):
        def func2():
            body
            return
        def func3(x):
            body
        func.func2 = func2
        func.func3 = func3
        return func
F=func(1)
F.func2()
F.func3(5)
```

Wrapper Function

- A function that takes another function as an argument
- Sub-function is to be defined in it which calls the function passed as an argument
- The function to be wrapped can written after wrapper decorator
  - E.g.

```
def func_decorator(func):
        def function_wrapper(arguments_optional):
                body
                func1()                 # function call, optional
                return_optional
        return function_wrapper
@func_decorator
def func1(arguments_optional):
        body
        return_optional
```

Questions