

Lambdas

- Lambdas are anonymous functions
- These are created inline using the following syntax:

lambda *<arguments>* : ***<expression>***

- Lambdas cannot span multiple lines
- Lambdas can only contain **expressions** and not **statements**
- No need of return statement in lambdas, as the value of expression is automatically returned
- **WAP** to create a lambda to return the square of a number.

Lambda Questions

- Create a lambda that returns the absolute value of a number : TODO
- Create a lambda to return sum of 2 numbers.
- Update the calculator to use a dictionary of lambda functions

Function and Scope

- The variable assignments done in a function create new objects that are local to the method

- Ex:

```
def method():  
    a = 10 # local  
    print(a)
```

```
method()  
print(a) # gives error
```

```
a = 0 # global
```

```
def funct0():  
    print(a)
```

```
def funct1():  
    a = 100  
    print(a)
```

The Global Keyword

- To access the variables at global scope, use the keyword **global**
- Ex:

```
a = 0 # global variable  
def funct():
```

```
    global a  
    print(a)  
    a = a+1  
    print(a)
```

```
funct()  
print(a)
```

```
# gives error; can't access local before declaring it  
a = 0
```

```
def funct():  
    print(a)  
    a = 100  
    print(a)
```

```
funct()
```

Nested Scope and Nonlocal Keyword (Python3)

- To access the variables at nested scope, use the keyword **nonlocal**

- Ex:

```
x = 0
def outer():
    x = 1
    def inner():
        x = 2
        print("inner:", x)
```

```
    inner()
    print("outer:", x)
```

```
outer()
print("global:", x)
```

```
x = 0
def outer():
    x = 1
    def inner():
        nonlocal x
        x = 2
        print("inner:", x)
```

```
    inner()
    print("outer:", x)
```

```
outer()
print("global:", x)
```

trainer.cpp@gmail.com

Sort method and lambdas

- Sorting a list of tuples containing name and age.

`[('Abhishek', '12'), ('Gaurav', 10), ('Rahul', '13'), ('Krishna', '11')]`

- Sort complete syntax:

`<list object> . sort(key= <some function>, reverse=False)`

<some function> should be a function taking a single argument and returning a single value (*a good candidate for a lambda*).

Higher Order Functions

- Functions that take functions as arguments or return functions are called higher order functions.
- **Map, reduce** and **filter** functions:
 - map(<function to apply>, <list of inputs>)
 - reduce(<function to apply>, <list of inputs>)
 - filter(<function to apply>, <list of inputs>)
- Available in **functools** module

MAP

- Map applies the function to each item of the iterable and returns a list containing the result of corresponding values.
- `L=[1,2,3,4,5]` WAP to create a list of square of these numbers
- Replace all spaces with `*` in a string.

Reduce

- `reduce(<function with 2 arguments >, <sequence type>)`
- Map applies the function to each item along with the result of the previous iteration
- So the function should take 2 arguments and return a single result.
- `L=[1,2,3,4,5]` WAP to find the sum of all the list elements

Filter

- Creates a list of elements for which a function returns true.
- So the function must be a **predicate Function**.
- $L=[1,2,3,4,5]$ WAP to create a list of only even numbers

Predicate Function

- A function that takes an argument and returns the **true** or **false** (a Boolean value) as a result.
- The **lambda** passed to the **Filter** function used in the previous case is Even Numbers example is a Predicate function.

Returning from function

- A function without a return statement return **None** by default
- The **None** return value is not printed on the interactive shell.

Something Simple : Fancy Generator and yield

- Generator is a method containing a **yield** statement.
- Generators can be used in **for** loops for iteration.
- Instead of a **return**, the **yield** stops the method when executed and returns the yield value.
- On next iteration, the next value is yielded on the basis of the function logic, continuing from the previous state
- WAG to replicate the range method.
- WAG to generate a string in reverse order

Recursive function

- A function that calls itself is a recursive function
- Printing first 10 natural numbers
- Finding sum of first **N** natural numbers

Functions and Arguments

- Default arguments

def funct(arg = value)

Provide a default value for missing arguments

- Variable length arguments

def funct(* args)

Passed arguments take the form of a tuple

- Keyword arguments

def funct(kwargs)**

Arguments are accepted as a dictionary

Decorator

- Decorators are function wrappers or simply **functions** taking **functions** as **arguments** and **returning functions**.
- Python provides a special syntax for using decorators, using the @syntax

@<name of decorator>

def <function name>(arguments): # normal definition of the function
 # code for the function

Seems like decorator

- ```
def decorator(func):
 print("Decorator")
 return func
```

 # decorator

```
def funct():
 print("Function")
```

 # function we will be decorating

```
f = decorator(funct)
print("After decorating")
f()
```

## Actual Decorator

---

- ```
def decorator(func):  
    def wrapper():  
        print("Decorator")  
        return func()  
    return wrapper
```

pass function to decorator
return whatever the function returns
return new function from decorator
- ```
def funct():
 print("Function")
```

# function we will be decorating
- ```
f = decorator(funct)  
print("After decorating")  
f()
```

Decorator syntax

- *def <decorator function name>(wrapped function arguments):
 def <wrapper name>(*args, **kwargs):
 # some operation involving function argument
 return <wrapped function>(*args, **kwargs)
 return <wrapper name>*
- Decorate a function to print execution time of a function
- Write a decorator to call a function **n** times