Second Order Programming

Intro2CS – week 10B
Functions are values too

```python
>>> type(print)
<class 'builtin_function_or_method'>
>>> p = print
>>> p("hello")
hello
>>> p == print
True
>>> 
```
Functions are values too

```python
>>> import math
global f = math.sqrt
>>> f(4)
2.0
>>> type(f)
<class 'builtin_function_or_method'>
```
Functions are values too

```python
>>> def inc(x):
...    return x+1

>>> type(inc)
<class 'function'>
>>> f=inc
>>> f(5)
6
```
Functions can be parameters

```python
def print_hi():
    print("hi")

def print_hello():
    print("hello")

def do_twice(f):
    f()
    f()

>>> do_twice(print_hi)
hi
hi

>>> do_twice(print_hello)
hello
hello
```
def make_cat_sound():
    print("meow")

def make_dog_sound():
    print("roff")

def make_noise(sound):
    for i in range(10):
        sound()
Polymorphism

class Cat:
    def make_sound(self):
        print("meow")

class Dog:
    def make_sound(self):
        print("roff")

def make_noise(animal):
    for i in range(10):
        animal.make_sound()
Lambda notation

Function of

That returns

```python
>>> f = lambda x: 2*x
>>> f
<function <lambda> at 0x000000000383DEA0>
>>> f(5)
10
>>> op = lambda : print("hi")
>>> op()
hi
>>> 
```
Using Lambda’s

```python
>>> x = [7, 3, -5, -1]
>>> sorted(x)
[-5, -1, 3, 7]
>>> sorted(x, key = lambda x : x**2)
[-1, 3, -5, 7]
```
def map_array(f, a):
    for i in range(len(a)):
        a[i] = f(a[i])

>>> z = [11, 12, 13]
>>> def double(x) : return 2*x

>>> map_array(double, z)
>>> z
[22, 24, 26]

>>> map_array(lambda x : x+1, z)
>>> z
[23, 25, 27]
...
def filter_array(f, a):
    ans = []
    for x in a:
        if f(x):
            ans.append(x)
    return ans

>>> z = list(range(10))
>>> filter_array(lambda x : x%3==1, z)
[1, 4, 7]
>>>
def reduce_array(op, a):
    t = a[0]
    for x in a[1:]:
        t = op(t, x)
    return t

>>> reduce_array(lambda x, y: x+y, [47, 11, 42, 13])
113
>>>
Python’s map, filter, and reduce

```python
>>> list(map(lambda x: x*x, range(7)))
[0, 1, 4, 9, 16, 25, 36]
>>> list(filter(lambda x: x%2!=0, [7, 5, 4, 3, 2, 6, 1]))
[7, 5, 3, 1]
>>> import functools
>>> functools.reduce(lambda x, y: x*y, range(1, 6))
120
```
Hadoop (Map-Reduce)

• Leading simple paradigm for parallel programming
  – Mapping of the elements can be done in parallel
  – Reduction can also (partially) be done in parallel -- if op is associative
  – If a partial result is lost due to hardware failure, only the missing result needs to be re-computed

\[
\begin{align*}
1 + 2 + 3 + 4 + 5 + 6 &= 1 + 2 + 3 + 4 + 5 + 6 \\
\end{align*}
\]
class Webpage:
    def __init__(self, url, text):
        self.__url = url
        self.__text = text

    def get_text(self):
        return self.__text
    def get_url(self):
        return self.__url

    def match_quality(self, query):
        
        if query in self.__url: return 1.0
        if " " + query in self.__text: return 0.9
        if query in self.__text: return 0.8
        return 0

    def snippet(self, query):
        if query not in self.__text: start = 0
        else: start = max(self.__text.index(query)-10, 0)
        return self.__text[start: min(start+50, len(self.__text))]
import functools

def search(query, list_of_pages):
    result_from_page = lambda page: (page.match_quality(query), page.get_url(), page.snippet(query))

    all_results = map(result_from_page, list_of_pages)

    filtered_results = filter(lambda r: r[0]>0, all_results)

    sorted_results = sorted(filtered_results, key = lambda r: -r[0])

    printable_results = map(lambda r: r[1]+"("+r[2]+")", sorted_results)

    answer = functools.reduce(lambda l1, l2 : l1 + "\n" + l2, printable_results)

    return answer
def test_search():
    web = [Webpage("singers.com", "Adele's songs are not here"),
           Webpage("Adele.com", "Home page for Adele"),
           Webpage("Swift.com", "Home page for Taylor Swift"),
           Webpage("hello.com", "Hello song, as sung by Adele")]
    print(search("Adele", web))

Adele.com(page for Adele)
hello.com(s sung by Adele)
singers.com(Adele's songs are not here)
>>>
Intermediate Prints

all_results = [(0.8, 'singers.com', 'Adele's songs are not here'), (1.0, 'Adele.com', 'page for Adele'), (0.0, 'Swift.com', 'Home page for Taylor Swift'), (0.9, 'hello.com', 's sung by Adele')]

filtered_results = [(0.8, 'singers.com', 'Adele's songs are not here'), (1.0, 'Adele.com', 'page for Adele'), (0.9, 'hello.com', 's sung by Adele')]

sorted_results = [(1.0, 'Adele.com', 'page for Adele'), (0.9, 'hello.com', 's sung by Adele'), (0.8, 'singers.com', 'Adele's songs are not here')]

printable_results = ['Adele.com (page for Adele)', 'hello.com (sung by Adele)', 'singers.com (Adele's songs are not here)']

Adele.com (page for Adele)
hello.com (sung by Adele)
singers.com (Adele's songs are not here)
...
def err_msg_presenter(level):
    if level == "novice":
        return lambda msg : "Problem -- don't worry"
    if level == "expert":
        return lambda msg : msg
    if level == "regular":
        return lambda msg : "Error: " + msg + " Look it up!"
    if level == "stubborn":
        return lambda msg : "!!!!" + msg + "!!!!" + msg + "!!!!"

presenter = err_msg_presenter("stubborn")
print(presenter("File Not found"))
Closure

def flat_tax_calculator(rate):
    return lambda amount : amount*rate

def threshold_tax_calculator(threshold, rate):
    def tax(amount):
        if amount < threshold:
            return 0
        return (amount-threshold)*rate
    return tax

tax_calc = threshold_tax_calculator(1000, 0.2)
# The tax rate above 1000 is 0.2
print("The tax on 5000 is", tax_calc(5000))
Currying

def minus(a, b):
    return a-b

def subtract(x):
    return lambda a : a-x

def new_minus(a, b):
    return subtract(b)(a)

print(subtract(2)(8))
print(new_minus(8,2))
Functions of functions

```python
def twice(f):
    return lambda x : f(f(x))

def f(x):
    return x+10

>>> g = twice(f)
>>> g(3)
23
>>> h = twice(g)
>>> h(4)
44
>>> four_times=twice(twice)
>>> q = four_times(f)
>>> q(7)
47
>>> twice(twice)(twice(f))(1)
```
Iterated functions

```python
def repeated(a, f, b):
    result = a
    for i in range(1, b):
        result = f(a, result)
    return result

def plus(a, b):
    return a+b

times = lambda a, b : repeated(a, plus, b)
power = lambda a, b : repeated(a, times, b)
tower = lambda a, b : repeated(a, power, b)

print("5+6=", plus(5, 6))
print("3*4=", times(3, 4))
print("2^5=", power(2, 5))
print("2^(2^(2^2))=", tower(2, 4))
```
Memoization

def memoize(f):
    memo = {}
    def helper(x):
        if x not in memo:
            memo[x] = f(x)
        return memo[x]
    return helper

def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

fib=memoize(fib)
print(fib(40))
Python Decorators

def memoize(f):
    memo = {}
    def helper(x):
        if x not in memo:
            memo[x] = f(x)
        return memo[x]
    return helper

@memoize
def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

print(fib(40))
def log_it(f):
    
def logging_f(x):
        print(f.__name__, "called on", x)
        v = f(x)
        print(f.__name__, "returned", v)
        return v
    
    return logging_f

@log_it
def factorial(n):
    if n==0: return 1
    return n*factorial(n-1)

>>> factorial(5)
factorial called on 5
factorial called on 4
factorial called on 3
factorial called on 2
factorial called on 1
factorial called on 0
factorial returned 1
factorial returned 1
factorial returned 2
factorial returned 6
factorial returned 24
factorial returned 120
factorial returned 120

>>>