Section 2

Introduction to Functional Programming in Python
Part 1

Section 2 - References

About Python Functional Programming

https://realpython.com/python-functional-programming/#:~:text=In%20functional%20programming%2C%20a%20program,advantages%20over%20other%20programming%20paradigms.

Why Functional Programming Matters

http://www.md.chalmers.se/~rjmh/Papers/whyfp.pdf

What is a Functional Language?

Opinions differ, and it is difficult to give a precise definition, but generally speaking:

- Functional programming is style of programming in which the primary method of computation is the application of functions to arguments;
- A functional language is one that supports and encourages the functional style.

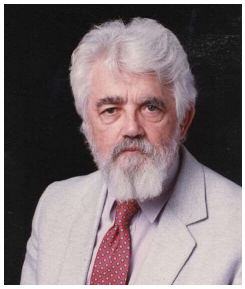
Historical Background

1920s - 1940s:

Alonzo Church and Haskell Curry develop the lambda calculus, a simple but powerful mathematical theory of functions.

Historical Background

1960s:



John McCarthy develops Lisp, the first functional language. Some influences from the lambda calculus, but still retained variable assignments.

Historical Background

1978:

John Backus publishes award winning article on FP, a functional language that emphasizes *higher-order functions* and *calculating with programs*.

Historical Background

1999:

The definition of Haskell 98 published, providing a long-awaited stable version of the language.

Principles- Functional Programs

- No Side effects
- No variable assignment
- A Function has no effect other than calculating its value

Key Features- Functional Programs

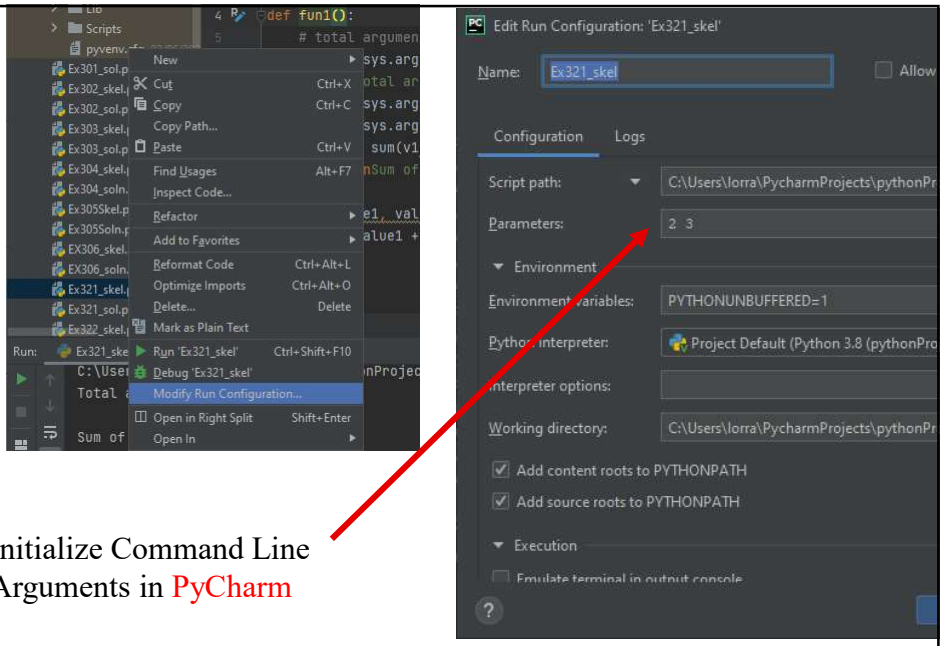
- Higher Order Functions
- Lazy Evaluation
- Currying
- Lambda Expressions
- List Processing

Functional Programs

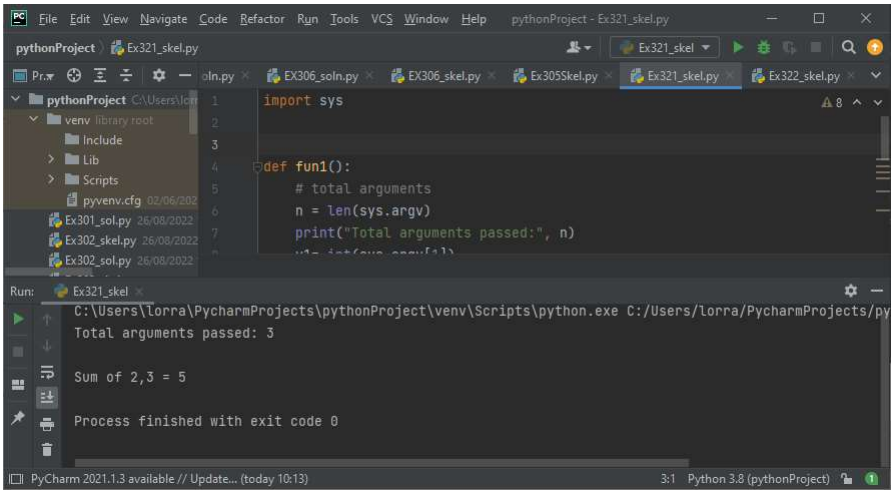
- Program consists only of Functions
- Main program is a function, receives input as parameters
- May in turn be defined as other functions

Functional Programs

```
import sys

def fun1():
    # total arguments
    n = len(sys.argv)
    print("Total arguments passed:", n)
    v1= int(sys.argv[1])
    v2= int(sys.argv[2])
    result = sum(v1,v2)
    print("\nSum of {0},{1} = {2}".format(v1,v2,result))

def sum(value1, value2):
    return value1 + value2

fun1()
```

Initialize Command Line Arguments in PyCharm



Running the Code

Ex321-

Modify so v1,v2,v3 are entered as Command-line arguments

```
def fun1():
    v1= 4
    v2= 6
    v3 = 14
    result = max(v1,v2,v3)
    print("\nmax of {0},{1},{2} = {3}".format(v1,v2,v3,result))

def max(value1, value2,value3):
    if (value1>value2 and value1 > value3):
        return value1
    elif (value2>value1 and value2 > value3):
        return value2
    else:
        return value3

fun1()
```

```
max of 4,6,14 = 14
```


- No side effects

```
def fun1():
    global result2
    v1=int(sys.argv[1])
    v2=int(sys.argv[2])
    result1 = sum(v1,v2)
    print("\n{0},{1}:  Sum = {2}, Diff = {3}".
                                format(v1,v2,result1,result2))

def sum(value1, value2):
    global result2
    result2 = value1 - value2 // side effect
    return value1 + value2
fun1()
```

```
7,2:  Sum = 9, Diff = 5
```

Functional Programming -

Function result only depends on input parameters

Always returns same result for a given parameters

Order of function calls irrelevant:

f(x)=2*x + 1 => f(2)=5 always

Next: A counter example

Sample call produces different results

Not Allowed in FP

```
def fun1():
   global v3
   v1=int(sys.argv[1])
   v2=int(sys.argv[2])
   v3=1
   result1 = add(v1,v2) # 1st Call
   v3=2
   result2 = add(v1,v2) # 2nd Call
   print("\n{0},{1}:  1st Call = {2}, 2nd = {3}“
                                .format(v1,v2,result1,result2))

def add(value1, value2):
   global v3
   return value1 + value2 + v3
fun1()
```

```
7,2:  1st Call = 10, 2nd = 11
```

Lazy Exaluation

•Postpose do operation until you are
•sure you need to do it

```
def fun1():
    target=int(sys.argv[1])
    list1 = [2,5,8,11]
    list2 = [1,5,9,11]
    result1 = search(target, list1)    # No Lazy Evaluation
    result2 = search(target, list2)
    if (result1 == True or result2 == True):
        print("\n {0} Found in Lists".format(target))
    else:
        print("\n {0} Not Found in Lists".format(target))

def search (target, list):
    print('Search')
    if (target in list):
        return True
    else:
        return False
```

```
Search
Search

 8 Found in Lists
```


Lazy Exaluation

```
def fun1():          # No Lazy Evaluation
    :
    result1 = search(target, list1)
    result2 = search(target, list2)
    if (result1 == True or result2 == True):
:
```

```
Search
Search

 8 Found in Lists
```

==========

```
def fun1():          # Lazy Evaluation
    :
if (search(target, list1) == True or search(target, list2) == True):
:
```

```
Search

 8 Found in Lists
```

No Side-effects - Summary

- No global variables
- No variable assignment
- Function just calculates a value based on input parameters

Next: Another counter example, variable assignment


```
def fun1():

    n=int(sys.argv[1])
    result1 = factorial(n)
    print("\n Factorial {0} = {1}:".format(n,result1))

def factorial(n):
    res=1
    for el in range(1,n+1):
        res = res * el    # variable reassignment
    return res
fun1()
```

```
Factorial 4 = 24:
```

Recursion

Most Functional Programming Languages use recursion

- avoids variable assignment

- facilitates Lazy Evaluation


```
def fun1():

    n=int(sys.argv[1])
    result1 = factorial(n)
    print("\n Factorial {0} = {1}:".format(n,result1))

def factorial(n):
    if (n==0):
        return 1
    else:
        return n * factorial (n-1)
fun1()
```

0! = 1
4! = 4 * 3!
n! = n * (n-1)!

```
Factorial 4 = 24:
```

Ex322 Rewrite using Recursion

```
def fun1():
    a = int(sys.argv[1])
    b = int(sys.argv[2])
    result1 = addInRange(a,b)
    print("Sum of Values {0} to {1}={2}:".format(a,b,result1))

def addInRange(first, last):
    result=0
    for el in range(first, last+1):
        result += el
    return result
```

```
Sum of Values 2 to 6=20:
```

2+3+4+5+6

addInRange(2,5) = 2 + addIntRange(3,5)
addInRange(3,2) = 0

Functional Programming

Two other key features

- List Processing
- Polymorphic Types

Recursion with Lists

```
def main():
  list1 = [2, 5, 2, 8]
  result = addList(list1)
  print('Sum of All Elements = {0}'.format(result))

def addList(listp):
  if len(listp)==0:
    return 0
  else:
      # remove first
    first=listp.pop(0)
    return first + addList(listp)

main()
```

addList[]=0
addList[2,5,2,8]= 2 + addList[5,2,8]

```
Sum of All Elements = 17
```

May 19 2011

27

Search

```
def main():
  list1 = [2, 5, 2, 8, 7, 3, 2, 5]
  target = int(input('Enter Target: '))
  result = searchTarget(list1, target)
  print('{0} Found in List = {1} '.format(target, result))

def searchTarget(listp, tar):
  if len(listp)==0:
    return False
  else:
    first=listp.pop(0) # remove first
    if (first==tar):
      return True
    else:
      return  searchTarget(listp,tar)
```

```
Enter Target: 6
Element 6 found in list = False
```

```
Enter Target: 8
Element 8 found in list = True
```

searchtTarget([],2) = False
searchTarget([2,5,6,1],2) = True
searchTarget([3,5,6,1],2) = searchTarget([5,6,1],2)

May 19 2011

28

List and Polymorphic Types

```
class Point:
    def __init__(self, a, b):
        self.__x = a
        self.__y = b

    def myPrint(self):
        print('(',self.__x, ',',
                    self.__y, ')')
----------------------------------------
class Name:
    def __init__(self, f, s):
        self.__first=f
        self.__surname=s

    def myPrint(self):
        print('Name=',self.__first,
                    self.__surname)
```

```
def fun1():
    list = [Name('John','Smith'),
            Point(2,3),
            Name('Peter','Shine')]
    printList(list)

def printList(list):
    for el in list:
        el.myPrint()
```

```
Name= John Smith
( 2 , 3 )
Name= Peter Shine
```


Recursion Using Dictionaries

```
import sys

class Student:
    def __init__(self, n,m):
        self.__name =  n
        self.__mark =  m

    def myPrint(self):
        print('Student=',self.__name,  self.__mark)

    def readName(self):
        return self.__name

    def readMark(self):
        return self.__mark
```

```
def fun1():
    s1=Student('Jones', 47)
    s2=Student('Shine', 66)
    s3=Student('White', 55)
    s4=Student('Smith',34)
    s5 = Student('Peters', 44)
    list = {1:s1, c 2:s2, 3:s3,  4:s4,  5:s5}

    target = input('Enter Name:')
    result = checkMark(list,target)   # return 0 if not in list
    if (result==0):
        print('{0} not in the list'.format(target))
    else:
        print('{0} Mark = {1}'.format(target, result))
```



```
def checkMark(list , target):
    if (len(list)==0):
        return 0
    else:
        student =  list.popitem()[1]
        if (student.readName()==target):
            return student.readMark()
        else:
            return checkMark(list,target)

fun1()
```

```
Enter Name:White
White Mark = 55
```

```
Enter Name:aa
aa not in the list
```

```
def fun1():
    s1=Student('Jones', 47)
    s2=Student('Shine', 66)
    s3=Student('White', 55)
    s4=Student('Smith',34)
    s5 = Student('Peters', 44)
    list = {1:s1, 2:s2,3:s3, 4:s4, 5:s5}
    result = sumAllMarks(list)   # return 0 if not in list
    print('Sum of All Marks = {0}'.format(result))
```

Ex323

Complete the Recursive Method ‘sumAllMarks’ to add all Student marks

```
Sum of All Marks = 246
```

33

33

Functional Programming -other key features

- Higher Order Functions
- Lazy Evaluation
- Currying
- Lambda expressions (Nameless functions)

Python Achieves this using lambda (nameless) functions

34

Lambda Functions

```
def main():
   val1 = int(input('Enter Value1: '))
   val2 = int(input('Enter Value2: '))
   val3 = int(input('Enter Value2: '))
   total = x(val1, val2,val3)
   print('Total=', total)

x = lambda a, b, c : a + b + c

main()
```

Nameless Functions

```
Enter Value1: 2
Enter Value2: 3
Enter Value2: 4
Total= 9
```

May 19 2011

35

```
def main():
   val1 = int(input('Enter Value1: '))
   val2 = int(input('Enter Value2: '))
   val3 = int(input('Enter Value2: '))
   total = calculate(x,val1, val2,val3)
   print('Total=', total)

x = lambda a, b, c : a * b * c

def calculate(f,a,b, c):
   return f(a,b,c)

main()
```

Higher Order Functions

```
Enter Value1: 2
Enter Value2: 3
Enter Value2: 4
Total= 24
```

May 19 2011

36

Normal Use of Lambda Functions

```
def main():
   val1 = int(input('Enter Value1: '))
   val2 = int(input('Enter Value2: '))
   val3 = int(input('Enter Value2: '))
   total = calculate( (lambda a, b, c : a - b - c) ,val1, val2,val3)
   print('Total=', total)

def calculate(f,a,b, c):
   return f(a,b,c)
main()
```

```
Enter Value1: 9
Enter Value2: 4
Enter Value2: 3
Total= 2
```

May 19 2011

37

Ex324 Complete the follow application

```
def main():
   val1 = int(input('Enter Value1: '))
   val2 = int(input('Enter Value2: '))
   val3 = int(input('Enter Value3: '))
   val4 = int(input('Enter Value4: '))
   result1 = hMax( pAdd ,val1, val2,val3,val4)
   print('Max of Added Pairs=', result1)
   result2 = hMax( pSub ,val1, val2,val3,val4)
   print('Max of Subtracted Pairs=', result2)

pAdd = lambda a, b : a + b

def hMax(f,a,b, c, d):
```

hMax(pAdd,5,1,6,4)
hMax (6, 10) = 10

hMax(pSub,5,1,6,4)
hMax (4, 2) = 2

```
Enter Value1: 5
Enter Value2: 1
Enter Value3: 6
Enter Value4: 4
Max of Added Pairs= 10
Max of Subtracted Pairs= 4
```

38

Built in Lambda fns in Python

```
# Program to filter out only the even items from a list
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list( filter(lambda x: (x%2 == 0) , my_list) )

print(new_list)
```

```
[4, 6, 8, 12]
```


Built in Lambda fns in Python

```
# Program to add 1 to each element of list
my_list = [1, 2,3,4,5,6]

new_list = list(map(lambda x: (x+1) , my_list))

print(my_list)
print(new_list)
```

```
[1, 2, 3, 4, 5, 6]
[2, 3, 4, 5, 6, 7]
```

Currying of Functions

```
# After stepping values by 3, list those < 6
my_list = [1,2,3,4,5,6,7,8,9]

new_list = list( filter(lambda x:(x<6),
                        list(map(lambda x: (x+3) , my_list))))
print(my_list)
print(new_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[4, 5]
```


Currying of Functions

```
# After stepping values by 3, list those < 6
my_list = [1,2,3,4,5,6,7,8,9]

new_list = list( filter(lambda x:(x<6),
                        list(map(lambda x: (x+3) , my_list))))
print(my_list)
print(new_list)
```

Output from this map is a list
This becomes an input to filter

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[4, 5]
```