FUNCTIONAL PROGRAMMING IN PYTHON:

AN INTRODUCTION

BY ALISHA ANEJA, PYCON INDONESIA 2017

ABOUT ME

M.Sc. (Computer Science), University of Melbourne

Software Developer Intern, Fedora

Machine Learning Intern, SilverPond

Contributor, Mozilla

WHAT IS FUNCTIONAL PROGRAMMING?

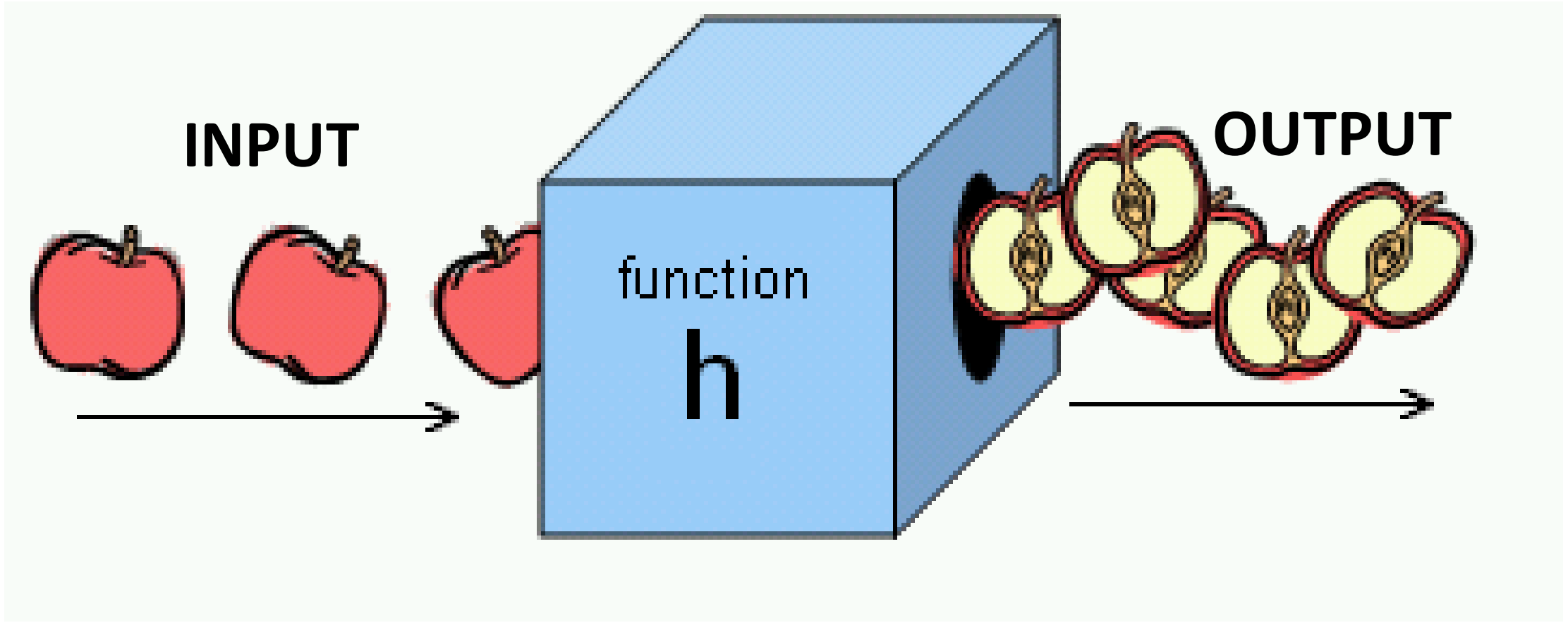
A programming paradigm!

WHAT IS FUNCTIONAL PROGRAMMING?

A way of approaching problems!

WHAT IS FUNCTIONAL PROGRAMMING?

Everything expressed as "functions"


INPUT
function
h
OUTPUT

```
str1 = "hello"
str2 = "world"
print(str1+" "+str2)
```

```
def hello():
    return “hello world”
```

FEATURES OF FUNCTIONAL PROGRAMMING

Pure functions, i.e. "no side effects"

Side effect of partying
till late....

SIDE-EFFECTS?

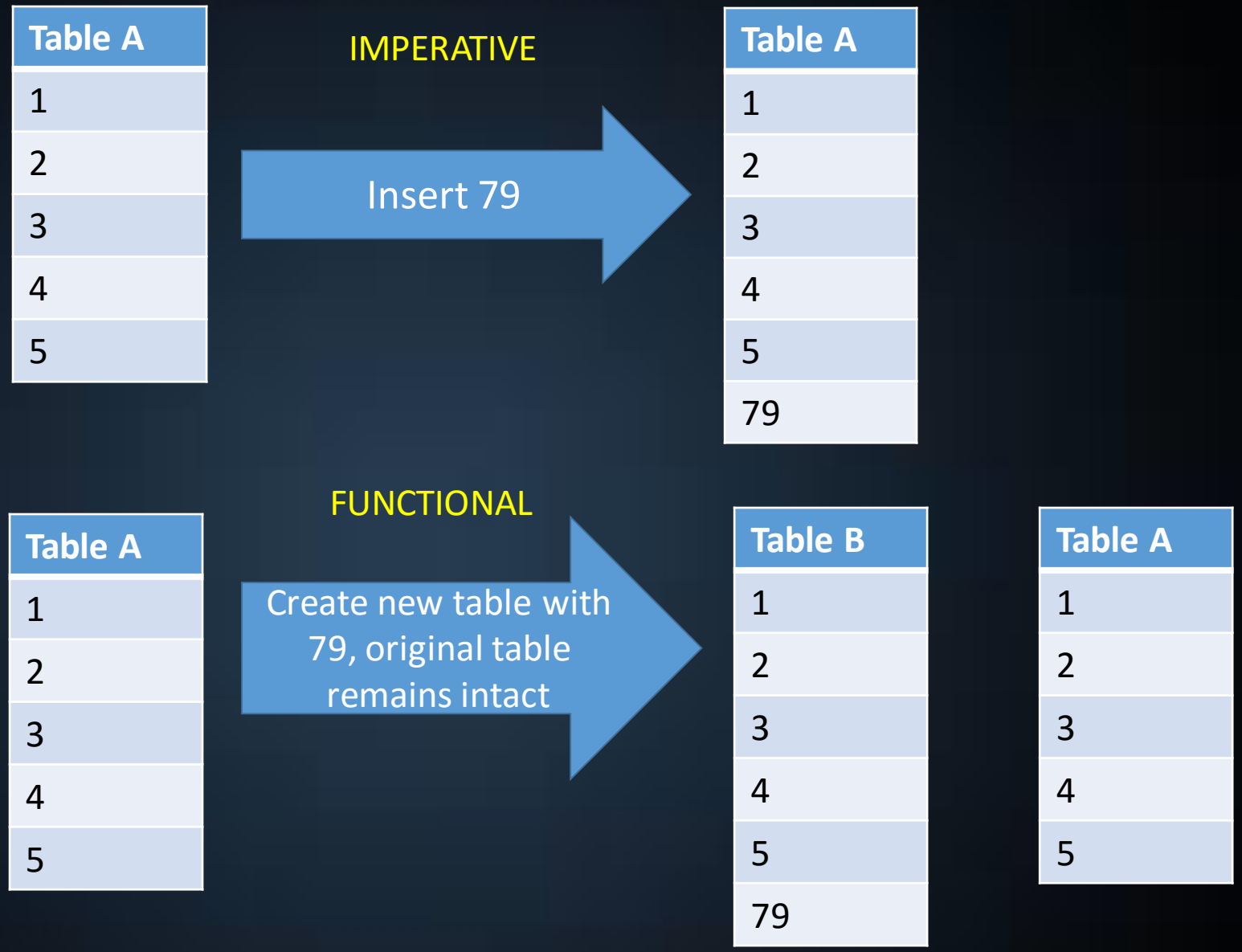
```
name = "Alisha Aneja"

def name():
    print("Hi, my name is", name)

name()
```

```
def name(name):
    return ("My name is %s"%name)

name("Alisha")
```

FEATURES OF FUNCTIONAL PROGRAMMING

Immutability

Destructive update



FEATURES OF FUNCTIONAL PROGRAMMING

Recursion instead of loops/iterations


TAIL CALL OPTIMIZATION
TO THE RESCUE

FEATURES OF FUNCTIONAL PROGRAMMING

Tail Call Optimisation

FEATURES OF FUNCTIONAL PROGRAMMING

Lazy Evaluation

FEATURES OF FUNCTIONAL PROGRAMMING

Partial Application

Haskell ->

```
add :: Int -> Int -> Int

add x y = x + y


addOne = add 1
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

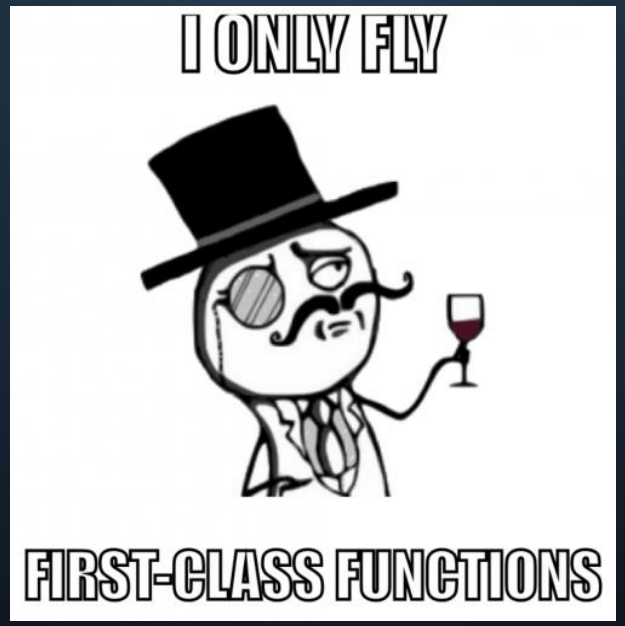
Immutable datatypes
(string, tuple/namedtuple, frozenset)

```
from collections import namedtuple

Person = namedtuple('Person', 'name gender')

alisha = Person(name = 'alisha', gender = 'f')

# Fields by name
print ("Name is %s and gender is %s."%(alisha.name, alisha.gender))

# Fields by index
print ("Name is %s and gender is %s."%(alisha[0], alisha[1]))

# Try mutating the elements (will get error)
alisha.name = "somerandomname"
```

```
food = ['cake', 'burger', 'pizza']
fSet = frozenset(food)

# Empty frozenset
print (frozenset())

# Try adding a new element: error
fSet.add('chocolate')

# Try removing a new element: error
fSet.remove('cake')
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

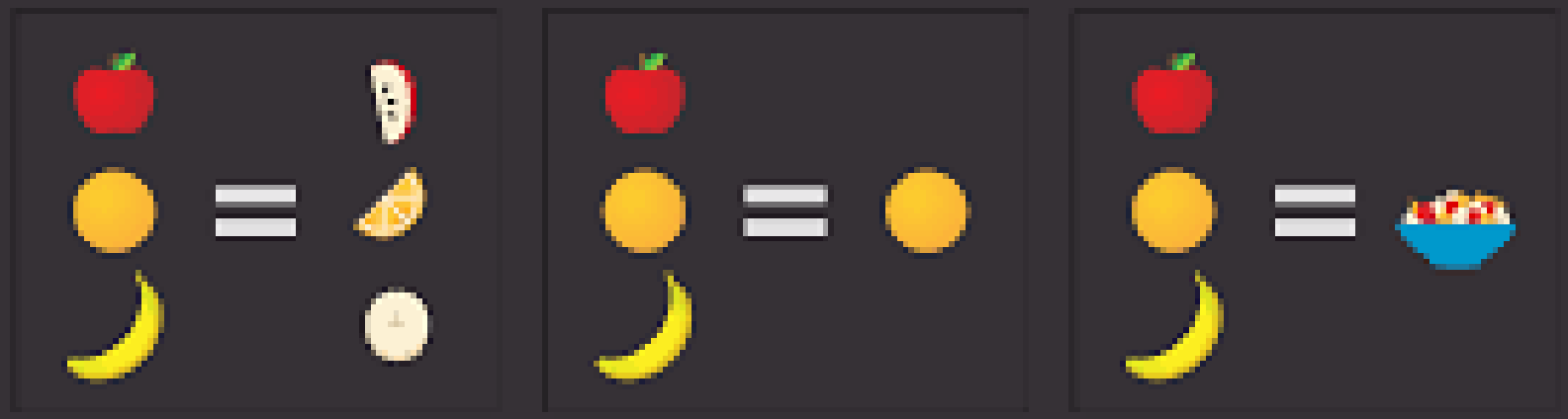
Functions as 'first class citizens'



```
def multiply2(a):
    return a*2

# Function assigned to variable
var_multiply = multiply2

# Function store in data structure (list here)
test_list = [multiply2, 7, 8]
print (test_list[0](3))

# Loop over list as with normal variables
for i in test_list:
    print (i)

# Function passed as argument to other functions
def add(multiply2, b):
    a = multiply2(3)
    return (a+b)

add(multiply2, 3)

# Return a function from another function
def greater(a, b):
    def yes_greater():
        return ("Oh yeah I am the bigger one! :)")
    def no_greater():
        return ("Oh no I am the smaller one! :(")

    if a > b:
        return yes_greater
    else:
        return no_greater

greater(5, 2)()
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

Higher order functions



map()

filter()

reduce()

```
def multiply2(a):
    return (a*2)

test_list = [3,5,6,7,8,9,11,12]

# will return [6, 10, 12, 14, 16, 18, 22, 24]
result = list(map(multiply2, test_list))

def greater_elem(a):
    if a > 10:
        return True
    else:
        return False

# will return [11,12]
result = list(filter(greater_elem, test_list))

# 282282
functools.reduce(lambda x,y: x*y, [47,11,42,13])
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

Lambda construct

```
test_list = [3,5,6,7,8,9,0]

# will return [6, 10, 12, 14, 16, 18, 0]
return_list = list(map(lambda x: x*2, test_list))
```

```
dict_a = [{'name': 'alisha', 'points': 10}, {'name':
'aneja', 'points': 8}]

# Output: ['alisha', 'aneja']
map(lambda x : x['name'], dict_a)
# Output: [100, 80]
map(lambda x : x['points']*10,  dict_a)
# Output: [True, False]
map(lambda x : x['name'] == "alisha", dict_a)
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

List comprehensions

```
test_list = [3,5,6,7,8,9,0]
mult = [x * 2 for x in test_list]

# In Haskell:
# let test_list = [3,5,6,7,8,9,0]
# [x*2 | x <- test_list]

# Combine the corresponding elements of two lists in pairs of tuples.
nums = [1, 2, 3, 4, 5]
letters = ['A', 'B', 'C', 'D', 'E']
nums_letters = [(n, l) for n in nums for l in letters]
print (nums_letters)
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

Iterators and generators
(lazy evaluation)

```
list1 = [2,4,7]

# iter() takes an iterable and gives one element at a time
a = iter(list1)

print (a.__next__()) # outputs 2
print (a.__next__()) # outputs 4
print (a.__next__()) # outputs 7
# print (a.__next__()) # StopIteration Exception

# Unpacking the iterator
list3 = [7,8,4]
x = iter(list3)
(a,b,c) = x # you should know the number of elements
print (a,b,c)
```

```
test_list = [2,3,5,8,8]

def elems(test_list):
    for i in test_list:
        yield i

a = yield_elems(test_list)
print (type(a)) # Type: <class 'generator'>

a.__next__() # 2
a.__next__() # 3
```

```
a = (x for x in [1,2,3,4,5] if x>2)

a.__next__() # 3
a.__next__() # 4
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

Module 'itertools'

```
list(chain([1,3,4,5], ['a','b','c'])) # [1,2,3,4,5,a,b,c]

list(cycle([1,3,4,5])) # [1,3,4,5,1,3,4,5,1,3,4,5.....]

list(islice([4,6,7,3], 1, 3, 2)) # [6]

list(combinations([1, 2, 3, 4, 5], 2))
list(combinations([1, 2, 3, 4, 5], 3))

list(permutations([1, 2, 3, 4, 5], 2))

list(takewhile(lambda x: x<5, [1,4,6,4,1]))

list(dropwhile(lambda x: x < 10, [1, 4, 6, 7, 11, 34, 66, 100, 1]))
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

Partial Application

functools.partial()

```
def buy(choice1, choice2, choice3):
    print("The food I like is %s, %s, %s"%(choice1, choice2, choice3))

order_sweet = partial(buy, choice3='ice cream')
order_sweet('chocolate', 'muffin') # The food I like is chocolate, muffin and ice cream
```

FUNCTIONAL PROGRAMMING IN PYTHON: HOW?

Recursion (be careful!)

WHAT IS MISSING IN PYTHON?

- No tail recursion optimization
- Mutable Variables
- Impossible to separate pure & non-pure functions
- No pattern matching (can use libraries though)
- No automatic partial application

ADVANTAGES OF FUNCTIONAL PROGRAMMING

Discuss!

LEARN MORE ABOUT FUNCTIONAL PROGRAMMING....

Libraries:
Toolz - "A functional standard library for Python"
funcy - "A fancy and practical functional tools"
hask - "Haskell language features and standard libraries in pure Python".
PyFunctional - "Python library for functional programming with collections in a data pipeline style".

Other resources:
https://github.com/sfermigier/awesome-functional-python
https://wiki.haskell.org/Tail_recursion/
http://learnyouahaskell.com/chapters

THANKYOU FOR YOUR ATTENTION!

ALISHA ANEJA
Email: anejaalisha37@gmail.com
Twitter: @alisha_aneja17
Github: alisha17
Website: https://alisha17.github.io

ANY QUESTIONS?