EC2401: INTRODUCTION TO DATA SCIENCE FUNCTIONAL THINKING

Evan Piermont

Problem of the Week

There are 9 coins, all except one are the same weight, the odd one is heavier than the rest. You must determine which is the odd one out using an old fashioned balance. What is the minimum number of weightings needed?

Recall, to create a function, the syntax is:

```
def funcName(x):

CODE THAT DEPENDS ON x

AND DETERMINES z

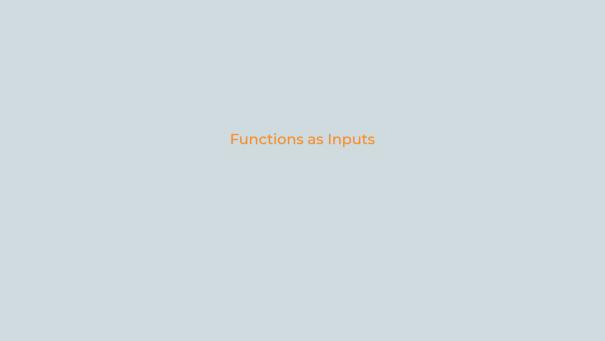
return z
```

Functions are Data

Functions can be manipulated just like any other data; function is an instance of the **Object** type.

You can:

- store a function in a variable.
- pass a function as a parameter to another function.
- return a function from a function.
- store functions in data structures: dicts, lists, ...



Functions can be called from within other functions and composed with on another. For example:

```
def square(x):
    return x*x
def halve(x):
    return x/2
x = square(halve(8))
y = halve(square(8))
```

will evaluate x = 16 and y = 32

```
def apply_twice(f,x):
    return f(f(x))
```

- The above takes a function and an argument and applies the function twice
- For instance, apply_twice(halve, 16) will output 4

There are some helpful built in Python functions that take functions as inputs.

- map applies a function to each element of an iterable
- **filter** extracts elements from an iterable for which a function returns True
- sorted sorts the elements of a given iterable in a specific order according to the output
- reduce apply a function to an iterable to reduce it to a single cumulative value

Map

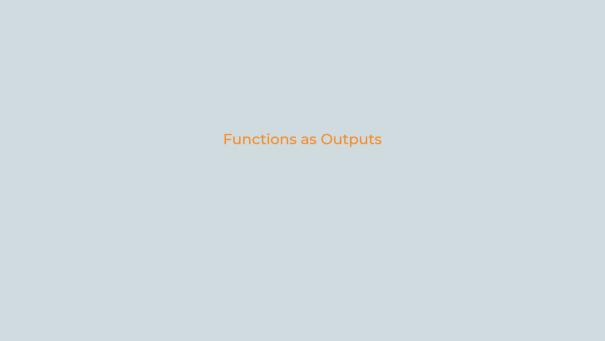
$$\begin{bmatrix} x, & y, & z \\ & \downarrow & \downarrow & \downarrow \\ & f(x), & f(y), & f(z) \end{bmatrix}$$

```
words = [`Another', `List', `Of', `Strings']
list(map(len, words))
```

- The above applies len, which returns the length of a string, to each element of 'words'
- ◆ The final line would output as [7,4,2,7]
- The whole thing is wrapped in the list function because 'map' actually returns a generator object

```
ns = [1, 2, 3, 4, 5, 6]
def even(n):
   if n % 2 == 0:
        return True
   return False
list(filter(even, ns))
```

- The above applies even to each element, and only keeps those that return True
 - Recall, many objects evaluate to True when converted to Booleans
- The final line would output as [2,4,6]



```
def gen_power(exp):
    def power(base):
        return base ** exp
    return power
```

- The above returns a function as its output
- We could define square = gen_power(2). Then square is a function
 - Same, as the function with the same name above
 - square(5) will output 25

Recursion

Recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem.

- In particular: by using functions that call themselves from within their own code.
- Essentially, allows for iteration within a function definition

 \bullet Perhaps we could solve for each n in order

- ullet Perhaps we could solve for each n in order
- We start with 1, then fact(1) is by definition 1

- ullet Perhaps we could solve for each n in order
- We start with I, then fact(1) is by definition I
- Then fact(2) is $1 \times 2 = fact(1) \times 2 = 2$

- ullet Perhaps we could solve for each n in order
- We start with 1, then fact(1) is by definition 1
- Then fact(2) is $1 \times 2 = fact(1) \times 2 = 2$
- Then fact(3) is $1 \times 2 \times 3 =$ fact(2) $\times 3 = 6$

- lacktriangle Perhaps we could solve for each n in order
- We start with 1, then fact(1) is by definition 1
 - Then fact(2) is $1 \times 2 = fact(1) \times 2 = 2$
 - Then fact(3) is $1 \times 2 \times 3 = fact(2) \times 3 = 6$
- Notice the pattern: $fact(n) = fact(n-1) \times n$

```
def fact(n):
    if n == 1:
        return 1
    return fact(n-1)*n
```

- The above applies exactly the logic from the last slide:
- If n = 1, return 1 by definition
- Otherwise, recursively call fact in the smaller problem

Here is a link to live examples.[©]