Functions in Python

A familiar function

```
import numpy as np

def manhattanDistance(coord1, coord2):
        dist = 0
        errorstring = "Coordinate dimension mismatch."
        if len(coord1)==len(coord2):
                for i in range(len(coord1)):
                        dist+=np.abs(coord1[i]-coord2[i])
                return dist
        else:
                raise runtimeError(errorstring)
```

A familiar function

```
def manhattanDistance(coord1, coord2):
    ...
    return dist
```

Let's focus on one part of that code snippet

- Defines a function called `manhattanDistance`, as well as its arguments
- Functions are extremely powerful!

Why Use Functions?

In Python (and most languages), we can just write code as a sequence of commands, and everything will be fine.

```
import numpy as np

myCoord1 = [10,20,30]
myCoord2 = [1,2,3]
dist = 0

dist+=np.abs(myCoord1[0]-myCoord2[0])
dist+=np.abs(myCoord1[1]-myCoord2[1])
dist+=np.abs(myCoord1[2]-myCoord2[2])
```

Why Use Functions?

What might go wrong with the code on the previous slide?

- What if I want to use a new set of coordinates with 4 dimensions?
- What if I don't notice that my coordinates do not have the same number of dimensions?
- What if I want to run that code as part of another program in a different file?
- What if I want to use Euclidean Distance in the future?

Nature of Functions

- Allow for reuse of code
    - Can import this code in other programs!
- Help us to organize our code
- Are limited in **scope** (more on that soon!)
- Allow us to quickly make broad changes

Starting to Write Functions

```
def myFunction(arguments_go_here):
```

First, we need to use the `def` statement to declare our function.

Later, after we have completed the code that runs inside of the function, we write our return statement:

```
        return objects_to_be_returned
```

Exercise

Write a function that returns the product of two numbers (note: the product of x and y is $x \times y$). Name the function `product`.

- What is the result of `product(2,5)`?
- What is the result of `product(2.71828,5)`?
- What is the result of `product("Howdy!",3)`?

Observations

When we use the function `product`, we are able to use a string as one argument. Why?

- Python is able to determine that the multiplier function `string * y` means that we want to repeat a string y times.

Exercise, Part 2!

Write a function that ONLY utilizes your `product` function to calculate the area of a circle with radius `r` (note: area is calculated as πr^2). Call that function `areaCircle`

- What is the result of `areaCircle(2)`?
- What is the result of `areaCircle(2.71828)`?
- What is the result of `areaCircle("Howdy!")`?

Observations

The function `areaCircle` can be created by utilizing our `product` function:

```
def areaCircle(r):
    r2 = product(r,r)
    return product(r2, 3.1415)
```

or, even more succinctly,

```
def areaCircle(r):
    return product(product(r,r), 3.1415)
```

Observations

- We can use functions inside of functions
- Use small functions to build part of a whole
- We can even use functions **recursively**

Recursive Functions

Recursive Functions

Try writing a function to calculate Fibonacci numbers.

$$F_0 = 0$$

$$F_1 = F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

How can we write a function to determine an arbitrary Fibonacci number?

Recursive Functions

```
def fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    elif n==2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

This function is **recursive** because it calls *itself* in order to complete its own execution.

Recursive Functions

Calling `fibonacci(5)` will trigger the following procedure:

- Because `n` is greater than 2, `fibonacci(n-1)` and `fibonacci(n-2)` are called
  - This would be `fibonacci(4)` and `fibonacci(3)`
- `fibonacci(4)` calls `fibonacci(3)` and `fibonacci(2)`
  - `fibonacci(2)` returns a value of `1`, and `fibonacci(3)` calls `fibonacci(2)` and `fibonacci(1)`, which both return values of `1`, causing `fibonacci(3)` to return a value of `2`
    ...

Recursive Functions

- The returned values of `fibonacci(3)` (`2`) and `fibonacci(2)` (`1`) lead `fibonacci(4)` to return a value of 3
- Remember that `fibonacci(3)` was also called by `fibonacci(5)` , and again will return a value of `2`
- `fibonacci(5)` thus returns the sum of 3 and 2 (the results of `fibonacci(4)` and `fibonacci(3)` , respectively)

Calling `fibonacci(5)` utilized the `fibonacci(n)` function 9 times, and did so **recursively**

Functions are the backbone of programming

We will use functions EVERY DAY as programmers, and they will save us a LOT of time as we move through more advanced topics.