FUNCTIONS
PART I

Objectives

- Describe what a function is and how they are useful
- Explain exactly what the return keyword does and some of the side effects when using it
- Add parameters to functions to output different data
- Define and diagram how scope works in a function
- Add keyword arguments to functions

What is a Function?

- A process for executing a task
- It can accept input and return an output
- Useful for executing similar procedures over and over

Why Use Functions?

- Stay **DRY** - **D**on't **R**epeat **Y**ourself!
- Clean up and prevent code duplication
- "Abstract away" code for other users
  - Imagine if you had to rewrite the "print()" function for every program you wrote

Function Structure

```
def name_of_function ():
    # block of runnable code
```

Our First Function!

```
def say_hi():
    print('Hi!')

say_hi()
# Hi
```

Another Function

```
def say_hi():
    'Hello!'

say_hi() # None
```

What's Wrong Here?

We can try to print, but what if we want to store the result of a function in a variable?

```
def say_hi():
    print('Hello!')

result = say_hi()

print(result) # None
```

Returning Values from Functions

```
def say_hi():
    return 'Hi!'

greeting = say_hi()

print(greeting) # 'Hi!'
```

`return`

- Exits the function
- Outputs whatever value is placed after the return keyword
- Pops the function off of the call stack

You can learn more here:

https://www.cs.ucsb.edu/~pconrad/cs8/topics.beta/theStack/02/

Yet Another Function

let's try some addition

```
def add(a,b):
    return a+b
```

What's **a** and **b**? Those are parameters!

Parameters

Variables that are passed to a function - think of them as placeholders that get assigned when you call the function.

```
def multiply(first, second):
    return first * second
```

you can call your parameters anything!

```
multiply(5,5) # 25
multiply(2,2) # 4
```

Common Return Mistakes

1. Returning too early in a loop

```
def sum_odd_numbers(numbers):
    total = 0
    for num in numbers:
        if num % 2 != 0:
            total += num
        return total
```

what's wrong here?

Common Return Mistakes

2. Unnecessary "else"

```
def is_odd_number(num):
    if num % 2 != 0:
        return True
    else:
        return False
```

```
def is_odd_number(num):
    if num % 2 != 0:
        return True
    return False
```

Naming Parameters

```
# Not great
def print_full_name(string1, string2):
    return(f"Your full name is {string1} {string2}")
```

```
# Better
def print_full_name(first_name, last_name):
    return(f"Your full name is {first_name} {last_name}")
```

Parameters vs Arguments

- A **parameter** is a variable in a method definition.
- When a method is called, the **arguments** are the data you pass into the method's **parameters**.
- **Parameter** is variable in the declaration of function.
- **Argument** is the actual value of this variable that gets passed to function.

YOUR TURN

Default Parameters

```
def add(a,b):
    return a+b

add() # does not work!
```

```
def add(a=10, b=20):
    return a+b

add() # 30
add(1,10) # 11
```

Default Parameters - Example

```
def show_information(first_name="Colt", is_instructor=False):
    if first_name == "Colt" and is_instructor:
        return "Welcome back instructor Colt!"
    elif first_name == "Colt":
        return "I really thought you were an instructor..."
    return f"Hello {first_name}!"

show_information() # "I really thought you were an instructor..."
show_information(is_instructor=True) # "Welcome back instructor Colt!"
show_information('Molly') # Hello Molly!
```

Why have Default Params?

Allows you to be more defensive

Avoids errors with incorrect parameters

More readable examples!

What can Default Parameters be?

Anything! Functions, lists, dictionaries, strings, booleans - all of the above!

```
def add(a,b):
    return a+b

def math(a,b, fn=add):
    return fn(a,b)

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

math(2,2) # 4

math(2,2, subtract) # 0
```

Just make sure they are the last parameters or you will get a SyntaxError!

Scope

Where our variables can be accessed!

Scope

Variables created in functions are scoped in that function!

```
instructor = 'Colt'

def say_hello():
    return f'Hello {instructor}'

say_hello() 'Hello Colt'
```

```
def say_hello():
    instructor = 'Colt'
    return f'Hello {instructor}'

say_hello()

print(instructor) # NameError
```

global

```
total = 0

def increment():
    total += 1
    return total

increment() # Error!
```

Lets us reference variables that were originally assigned on the global scope

```
total = 0

def increment():
    global total
    total += 1
    return total

increment() # 1
```

nonlocal

Lets us modify a parent's variables in a child (aka nested) function

```
def outer():
    count = 0
    def inner():
        nonlocal count
        count += 1
        return count
    return inner()
```

You will not find yourself using the global or nonlocal keyword frequently - but it's essential to understand for scope!

YOUR TURN

Keyword Arguments

```
def full_name(first, last):
    return "Your name is {first} {last}"

full_name(first='Colt', last='Steele') # Your name is Colt Steele

full_name(last='Steele', first='Colt') # Your name is Colt Steele
```

Order does not matter!

Why use Keyword Arguments?

You may not see the value now, but it's useful when passing a dictionary to a function and unpacking it's values - we'll see that later!

A little more flexibility

Different from Default Params

When you define a function and use an =
you are setting a default parameter

When you invoke a function and use an =
you are making a keyword argument

Example

```
def full_name(first="Colt", last="Steele"):
    return "Your name is {first} {last}"

full_name() # Your name is Colt Steele

full_name(last='Enthusiast', first='Python') # Your name is Python Enthusiast
```

Documenting functions

Use """ """

Essential when writing complex functions

```
def say_hello():
    """A simple function that returns the string hello"""
    return "Hello!"

say_hello.__doc__ # 'A simple function that returns the string hello'
```

Recap

- functions are procedures for executing code. They accept inputs and return outputs when the return keyword is used
- To create inputs, we make parameters which can have default values, we call those default parameters
- variables defined inside of functions are scoped to that function - watch out for that!
- When invoking a function we can pass in keyword arguments in any order, we'll see this more later!
- Be careful to not return too early in your conditional logic and refactor when you can to remove unnecessary conditional logic. Make sure you don't return in a loop too early as well!

YOUR
TURN