

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light green color. They are positioned diagonally, with the blue one in front of the green one.

# Advanced Python

Research Data Services  
Avery Fernandez



# Lesson Plan

- Define custom functions
- Create classes
- Import custom libraries
- Create Script Files



# Functions

A function is instructions that run only once it is called.

3 core features of a function:

- Pass it arguments and data
- Do operations on the data given
- Returns back data

<https://docs.python.org/3/tutorial/controlflow.html#defining-functions>



# First Function

Syntax: `def functionName(args):`

A function is declared by the `def` statement

The `functionName` is just a placeholder name, it can be anything

The args can just be anything you wish to pass through



# First Function

```
def printName():  
    print("John")
```

Won't run until we call the function

```
printName()
```

```
John
```



# Arguments

```
def printName(name):  
    print(name)
```

```
printName("Johnny")
```

What is happening

```
def printName(name="Johnny"):  
    print(name)
```

Johnny



# Return data

```
def helloSentence(name):  
    return "Hello, " + name
```

```
sentence = helloSentence("Robert")  
  
print(sentence)
```

The return statement just means that it gives back a value for you to use later.

```
"Hello, Robert"
```

```
Hello, Robert
```



# Function Arguments

```
def adding(n1, n2):  
    return n1+n2  
  
print(adding(1))
```

```
def adding(n1=0, n2=0):  
    return n1+n2  
  
print(adding(1))
```

Must have the same amount of arguments passed as specified in the function

```
TypeError: adding() missing 1  
required positional argument: 'n2'
```

Unless the variables are already “initialized”

1





# Function Arguments

The \* means grab whatever is left over

```
def children(*kids):  
    print("First child is", kids[0])  
  
    print("Last children are",  
kids[1:-1])  
  
    print("The rest are", kids[-1])  
  
children("Kenny", "Jimmy", "Smith",  
"Keath")
```

First child is Kenny

Last children are ('Jimmy', 'Smith')

The rest are Keath



# Function Argument Order

```
def foods(fruit, vegetable, drink):
```

```
    print(fruit, vegetable, drink)
```

```
foods("lettuce", "sprite", "apple")
```

```
foods(vegetable="lettuce",  
      drink="sprite", fruit="apple")
```

Order doesn't matter if you specify the variable

```
lettuce sprite apple
```

```
apple lettuce sprite
```



# Function values as dictionary

```
def boats(**boat):
```

```
    print(boat["speed"])
```

```
Fast
```

```
boats(speed = "Fast")
```



# Function Values as Dictionary

```
def boats(**boat):
```

vroom

```
    if "cargo" in boat.keys():
```

```
        print(boat["cargo"])
```

```
    if "speed" in boat.keys():
```

lots of stuff

```
        print(boat["speed"])
```

fast boi

```
boats(speed="vroom")
```

```
boats(cargo="lots of stuff",  
      speed="fast boi")
```



# Function Return

```
def fruits(fruit1, fruit2, fruit3):  
    return fruit1, fruit2, fruit3
```

```
myFruits = fruits("apple", "orange",  
"kiwi")  
  
print(myFruits)
```

Data is returned as a tuple

```
('apple', 'orange', 'kiwi')
```



# Function Return

```
def fruits(fruit1, fruit2, fruit3):  
    return fruit1, fruit2, fruit3
```

```
firstFruit, secondFruit, thirdFruit =  
fruits("apple", "orange", "kiwi")  
  
print(firstFruit)  
  
print(secondFruit)  
  
print(thirdFruit)
```

Split the data based on the tuple

apple

orange

kiwi



# Function Return

```
def fruits(fruit1, fruit2, fruit3):  
    return fruit1, fruit2, fruit3
```

```
firstFruit, *leftoverFruit =  
fruits("apple", "orange", "kiwi")  
  
print(firstFruit)  
  
print(leftoverFruit)
```

Can even use the \* symbol to grab the leftovers

```
apple  
['orange', 'kiwi']
```



# Pass Function

```
def notFinished() :
```

```
    pass
```

```
notFinished()
```

Use the pass statement if you wish to do nothing inside the function





# Exercise 1

Create a function that takes 5 numbers

It multiplies together the first and last number

It adds together the middle 3 numbers

Returns the sum, then the product

Then print the sum and product outside the function



# Classes

Classes allow you to apply attributes over a large range of things

If you have 10 students, each with a hair color, birthday, height, and grade

Creating Variables for each one would be a pain

<https://docs.python.org/3/tutorial/classes.html>



# First Class

```
class food:
```

```
    fruit = "apple"
```

Creates a class called food and has an attribute called fruit that stores apple

```
myFood = food()
```

```
print(myFood.fruit)
```

```
apple
```

Applies all the attributes to myFood



# Changing Attributes

```
class food:
```

```
    fruit = "apple"
```

Easily change attributes

```
myFood = food()
```

```
print(myFood.fruit)
```

apple

```
myFood.fruit="orange"
```

```
print(myFood.fruit)
```

orange



# Class Initializers

Syntax: `def __init__(self, args):`

Allows us to initialize attributes with custom values



# Customize Values

```
class food:

    def __init__(self, fruit, vegetable,
drink):

    self.fruit = fruit

    self.vegetable = vegetable

    self.drink = drink

myFood = food("apple", "lettuce", "coke")

print(myFood.drink)
```

myFood now has the custom values for fruit, vegetable,  
and coke

Uses function notation

```
apple lettuce coke
```



# Class Print Statement

```
class food:

    def __init__(self, fruit, vegetable, drink):

        self.fruit = fruit

        self.vegetable = vegetable

        self.drink = drink

    def __str__(self):

        return self.fruit + " " + self.vegetable +
" " + self.drink
```

```
myFood = food("apple", "lettuce", "coke")
```

```
print(myFood)
```

```
apple lettuce coke
```



# Class Methods

```
class food:

    def __init__(self, fruit, vegetable, drink):

        self.fruit = fruit

        self.vegetable = vegetable

        self.drink = drink

    def myMethod(self):

        print("My favorite fruit is",
self.fruit)

myFood = food("apple", "lettuce", "coke")

myFood.myMethod()
```

My favorite fruit is apple





## Exercise 2

Create a class to store student data

The class must store the name, age, height, major

Create a print state that outputs, "My name is ... and I am studying ...."

Create a method that multiplies the age by their height



# Custom Libraries

Custom Libraries allow you to create functions that can be used in almost any python file

Custom Libraries are just regular python files that you can import



# mathematics.py

```
def summation(*numbers):  
  
    total = 0  
  
    for number in numbers:  
  
        total+=number  
  
    return total
```



# Using custom library

```
import mathematics
```

55

```
print(mathematics.summation(1,2,3  
,4,5,6,7,8,9,10))
```



# Creating Script Files

testing.py

```
import custom

if __name__ == "__main__":

    custom.printHello()
```

customs.py

```
def printHello():

    print("Hello World")

if __name__ == "__main__":

    for i in range(1000):

        print(i)
```

Will only run if the file is the one that is running

[https://docs.python.org/3/library/\\_\\_main\\_\\_.html](https://docs.python.org/3/library/__main__.html)



## Exercise 3

Create a custom library with 2 functions

Function 1: Returns the product of a bunch of numbers

Function 2: prints the given string out

Create a script file that imports the library then uses both functions



# NOTES

