Class 5 - Functions

[w200] MIDS Python for Data Science Summer 2018

Course Content | First 8 Weeks - Programming

Unit 1 | Introduction, the Command Line, Source Control

Unit 2 | Starting Out with Python

Unit 3 | Sequence Types and Dictionaries

Unit 4 | More About Control and Algorithms

Unit 5 | Functions

Unit 6 | Modules and Packages

Unit 7 | Classes

Unit 8 | Object-Oriented Programming

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

Special Arguments - Activity 1

Recursion

Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

Week 5 | Course Schedule

Course Schedule:
https://docs.google.com/spreadsheets/d/11DxadnNwyFaJIPYLUJSPUlNGCtTenBCR4yaR1CbFBKg

Week 5 | Homework 4

1. For loop
2. Chess
3. Algorithms - Binary search, discussion and fix
4. Comprehensions

Discuss: What was the hardest part of HW4?

Poll: How long did you spend on this week’s assignment?

Assignment Review | Week 3

Refresher:

1. Pig Latin
2. Matrix Inverter
3. To-do List
4. Fibonacci Series
5. Pascal’s Triangle

Assignment Logistics | Week 3

- As far as formatting / design decisions with your submission:
  - Please write in code comments at the top why you decided to format / design your answer that way (using # or ‘’’ docstrings)
  - If that choice does not contradict with the requirements of the question and makes some sense - no points will be deducted
  - Please use common sense on what the code needs to do; if you were the user what would you expect out of this code?

Assignment Review | Week 3

- **Grading questions: Please email all four of us (Chris, Gunnar, Gerry & Rob) so we can discuss!**
- Test your code thoroughly to avoid crashes - run it with ‘edge’ cases!
- Check for incorrect math and formulas! Might need to do them by hand first and see if the code matches.
- RTQ! If the question states ‘<=’ then make the code do that

Assignment Review | Week 3 (cont)

- Python Spacing:
  - Indents are 4 spaces or 1 tab
  - Please put empty lines before blocks of code for readability!
  - In Python, there are usually spaces before and after operands
    (Eg. a = b + c rather than a=b+c)
- Comments:
  - Please put comments on their own line instead of ‘squishing’ them in at the end of the line
  - Please delete the debugging statements (e.g. commented out print statements) when you turn in your final code

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

Special Arguments - Activity 1

Recursion

Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

Git | Branching and Merging

Make the branch

```
git checkout -b  <name>     # to add a new branch
git branch                  # tells you which branches there are.
```

merge into the branch

```
git checkout master     #change into the branch that you want to merge into
git merge <alternative branch>   #”fast forward” merge indicates no conflicts.
```

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

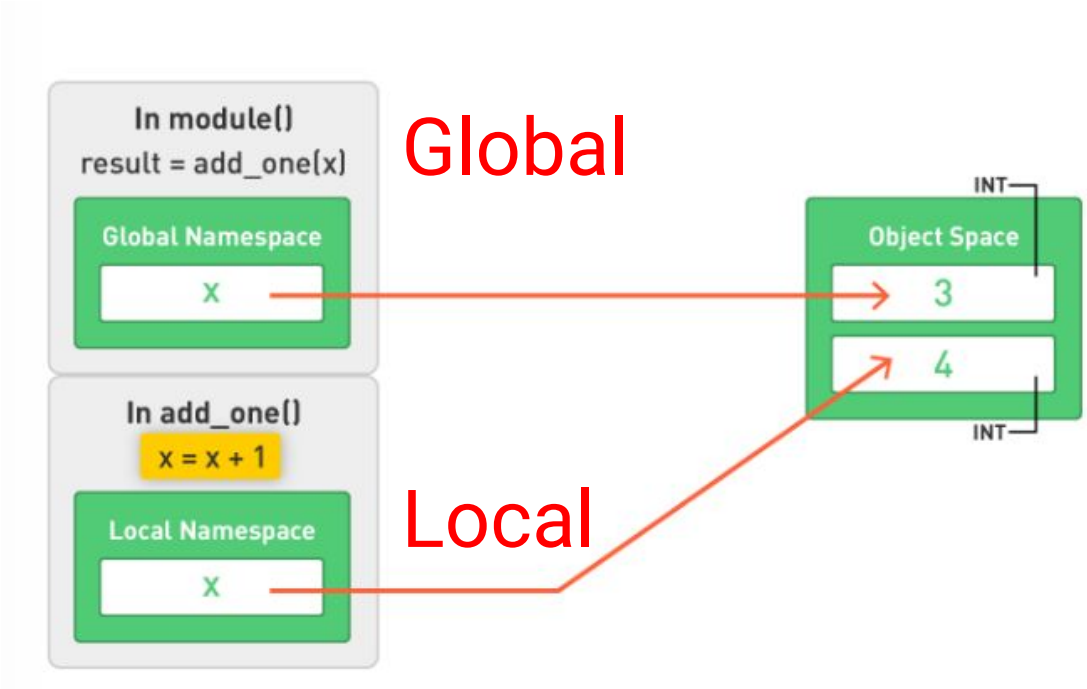
Special Arguments - Activity 1

Recursion

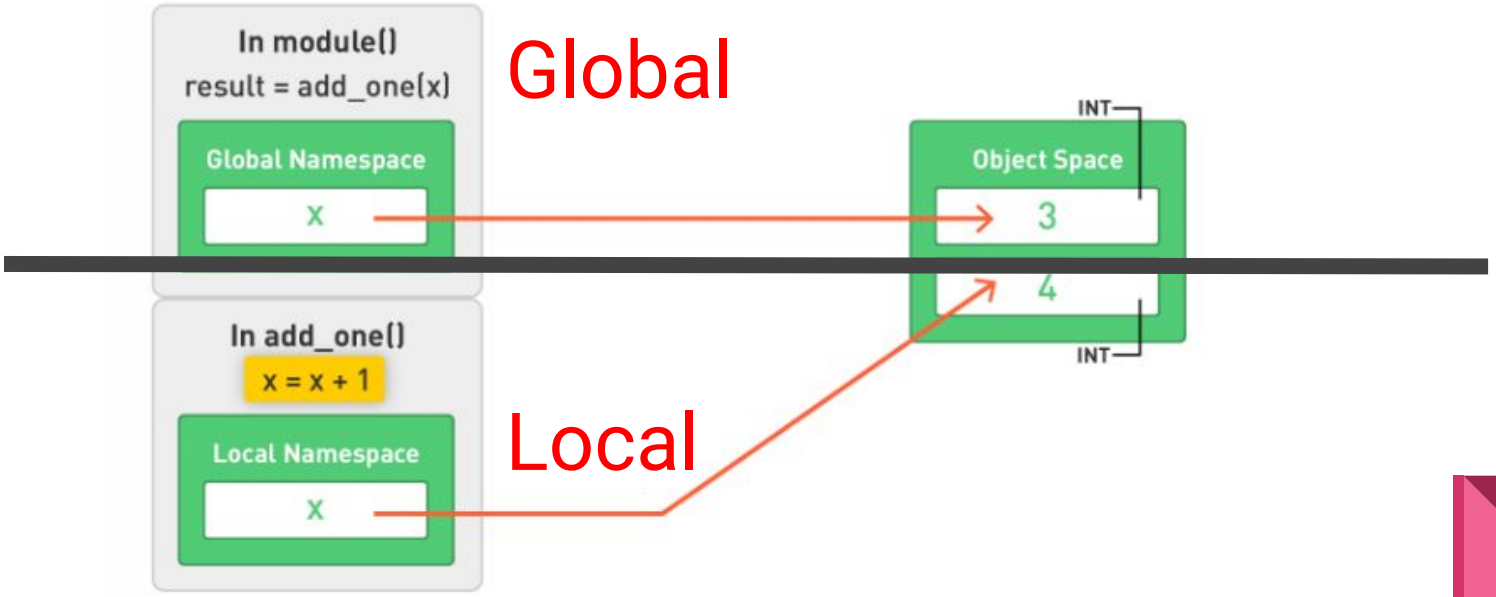
Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

Namespace | Global vs. Local



Namespace | Global vs. Local



Namespace | Global vs. Local

- Why have separate namespaces?
  - What are local variables used for (examples)
  - What are globals used for (examples)

- What happens when a variable is called but not defined in the local namespace?

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

Special Arguments - Activity 1

Recursion

Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

Functions | Anatomy

- Arguments when called, parameters once inside.
- Docstring # help text
  - Readability, reusability
- Code reuse, modularity, abstraction
- Function is object
- Function is not executed until it is called
- Return statement

```
##Square root algorithm as a function
def sqrt(x, epsilon):
    """Newton's Method to find square root
       with precision epsilon (Heron's algorithm)"""
    ans = 1
    num_guesses = 0
    while abs(x/ans - ans) > epsilon:
        ans = (x/ans + ans)/2
        num_guesses += 1
    return ans
```

Functions | Flow through the Function

1- Define function
2- Execute function with arguments
3- Arguments become parameters inside function
4- Internal operations
5- Return variable
6- Assignment to variable outside the function

```
def distance_to_origin(x, y):
    """find the distance from a point at (x, y) to the origin"""
    ans = sqrt(x**2 + y**2, 0.00001)
    return ans

magnitude = distance_to_origin(x, y)
```

Functions | as objects

```
type(round10)
```

```
function
```

- It has a type

```
a = round10
a(12)
```

```
10
```

- Can bind it to a variable name

- Can be passed as an argument
  - Perform operation (function) on iterable (grade list)

```
def apply_to_grades(operation, grade_list):
    return [(name, operation(grade)) for name, grade in grade_list]

grade_list = [("Betty", 88), ("Steve", 75), ("Bob", 73), ("Teri", 94), ("Sandy", 97)]
print( apply_to_grades(round10, grade_list))
```

Functions | map and lambda

- map()
  - apply function to each element of an iterable

```
list(map(round10, (23, 45, 42, 66)))
```

```
[20, 50, 40, 70]
```

- Lambda functions, “anonymous”
  - just means we don’t bother to name them
  - pass lambda as an argument

```
lambda x : 100 - (100 - x)/2
```

```
<function __main__.<lambda>>
```

```
apply_to_grades(lambda x : 100 - (100 - x)/2, grade_list)
```

```
[('Betty', 94.0),
 ('Steve', 87.5),
 ('Bob', 86.5),
 ('Teri', 97.0),
 ('Sandy', 98.5)]
```

Functions | map and lambda

- It is common to combine **map** and **lambda**

```
list(map(lambda x: x**2, (23, 45, 42, 66)))
```

```
[529, 2025, 1764, 4356]
```

Function

Iterable

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

Special Arguments - Activity 1

Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

Special Arguments | flexibility

- **What does it mean for code to be “Brittle” or “Fragile”?**
- **Special methods increase flexibility**
  - Default values
  - None type

```python
def feedback(grade=None, comment=None):
    text = "" if comment == None else " - " + comment

    if grade == None:
        return("Grade is Missing. " + text)
    elif grade >= 90 and grade <= 100:
        return("A " + text)
    elif grade >= 80 and grade < 90:
        return("B " + text)
    elif grade >= 70 and grade < 80:
        return("C " + text)
    elif grade >=60 and grade < 70:
        return("D " + text)
    else:
        return("F " + text)
```

```python
print(feedback(80))
```

```
B
```

```python
print(feedback(75, "Please study more"))
```

```
C  - Please study more
```

```python
print(feedback())
```

```
Grade is Missing.
```

Default Values | modified over time

- Default value won't be reset if called a second time!
- "Permanent" attributes of function (in the global namespace)
- Modified once used

```
def add_total(order_list = []):
    total = sum([quantity for name, quantity in order_list])
    order_list.append( ("Total", total) )
    print(order_list)
```

```
add_total()
```

```
[('Total', 0)]
```

```
add_total()
```

```
[('Total', 0), ('Total', 0)]
```

Keyword Arguments | make it clear

- Arguments usually use positional cues
  - We can define in any order if we specify keywords

```
print(feedback(90, comment="Keep it up!"))
```

```
A  - Keep it up!
```

```
print(feedback(comment="Not bad.", grade=88))
```

```
B  - Not bad.
```

Arguments | from CLI (command line)

Sys.argv

- When you pass arguments into a .py script on the command line
- They can be accessed using sys.argv
- Shows how you pass in a variable into a python script

```
import sys

print(sys.argv)

if len(sys.argv) > 1:
    name = sys.argv[1]
else:
    name = input("Enter your name: ")

for i in range(len(name), 0, -1):
    print( name[0:i], end = " ")
    for j in range(i, len(name)):
        print(" " * (j-i) + name[j], end="")
    print("")
```

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

Special Arguments - Activity 1

Recursion

Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

Recursion | Overview

Question: What is recursion?

Recursion | Base Case and Recursive Rule

Question: What is recursion?

Answer: Recursion occurs whenever a function calls itself. It is a programming technique that requires you to specify:

1. A base case
2. A recursive rule

The “base case” ends the process of the recursive rule (loop) calling itself.

Recursion | Base Case and Recursive Rule

Consider the code below. What is the base case, and what is the recursive rule? Why do we need each?

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

Recursion | Base Case and Recursive Rule

Consider the code below. What is the base case, and what is the recursive rule? Why do we need each?

Base case

Recursive Rule

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

Recursion | Recursion vs. Looping

Question: Why would you use recursion instead of a loop?

Recursion | Recursion vs. Looping

Question: Why would you use recursion instead of a loop?

Answer: Recursion allows you to know even “less” about the structure of a problem in advance. You can traverse interesting data structures like JSON and trees.

We discussed that a “while” loop allows you to run a single loop an unknown (in advance) number of times. Recursion allows you to run an unknown number of loops an unknown number of times.

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

Special Arguments - Activity 1

Recursion

Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

Error Checking | fail gracefully

- Try/except,
  - Except can be specific
  - Except can be general

```
try:
    x = float(input("Enter a number: "))
    print("The reciprocal of your number is", 1/x)
except ValueError:
    print("You did not enter a valid number")
except ZeroDivisionError:
    print("Zero does not have a reciprocal")
except:
    print("Something else went wrong")
```

Raising and Capturing Errors | “as e”

- Raise exception and capture it
  - Higher level functions can capture the exception
    i. and bind it “as e”
  - Lower level function creates exception
    i. use **raise**

```
try:
    sell("oranges", 15, inventory)
except Exception as e:
    print("Could not complete sale: " + str(e))
```

```
Could not complete sale: Inventory does not contain enough of item to sell.
```

```
def sell(item, quantity, inventory):
    if item not in inventory:
        raise Exception(str(item) + " does not appear in inventory.")
    q = inventory[item]
    if q < quantity:
        raise Exception("Inventory does not contain enough of item to sell.")
    inventory[item] = q - quantity
```

Week 5 | Agenda

Week 4 Assignment - High Level Discussion and Polls

Git Branching and Merging

Namespaces: Global vs. Local Variables

Functions

Special Arguments - Activity 1

Recursion

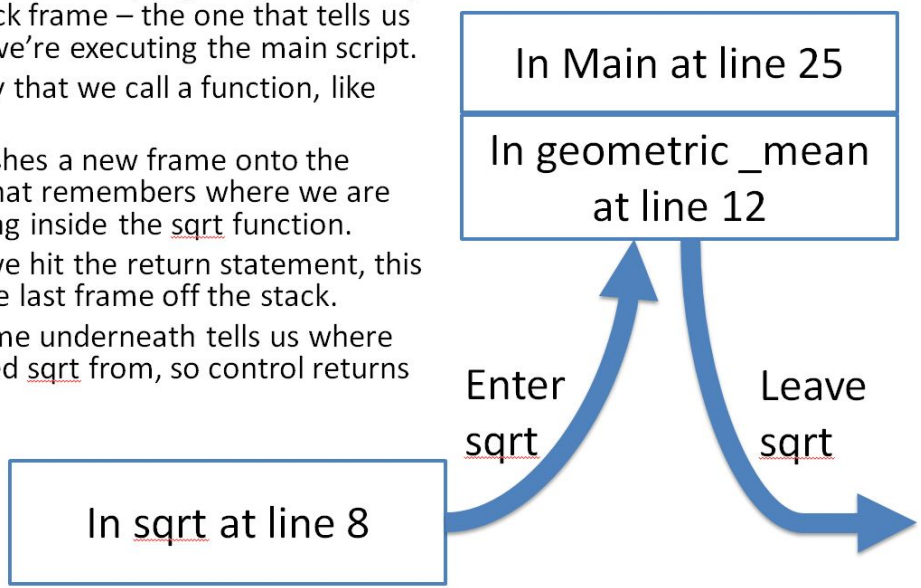
Error Handling - Activity 2

The Call Stack and Stack Trace - Activity 3

The Stack Trace | Visual Understanding

Call Stack

- When you start a program, there's only one stack frame – the one that tells us where we're executing the main script.
- Let's say that we call a function, like sqrt.
- This pushes a new frame onto the stack, that remembers where we are executing inside the sqrt function.
- When we hit the return statement, this pops the last frame off the stack.
- The frame underneath tells us where we called sqrt from, so control returns there.



The stack - try recursion in python tutor

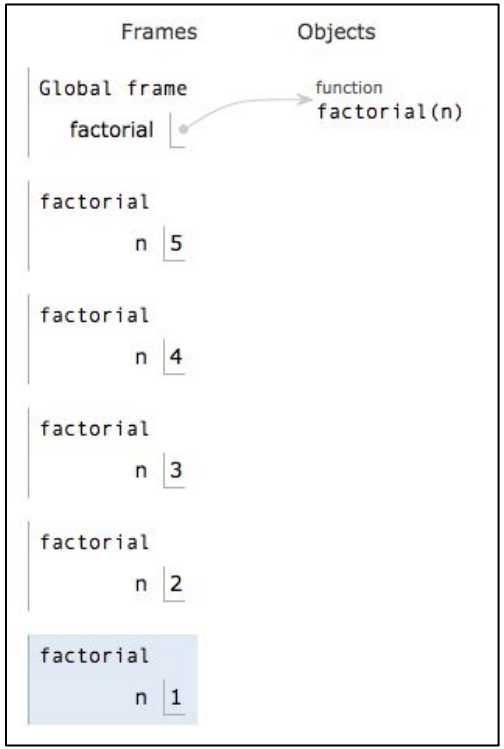
Python 2.7

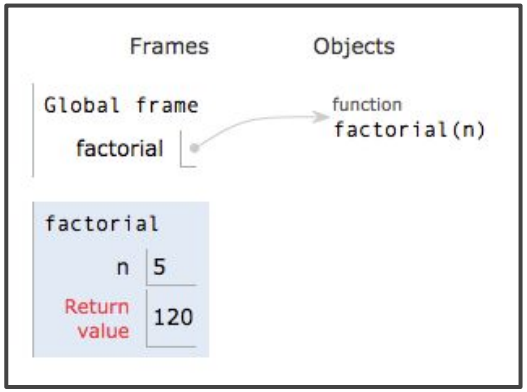
```
1 def factorial(n):
2     if n == 1:
3         return 1
4     return n * factorial (n-1)
5
6 factorial(5)
```

Edit code | Live programming

line that has just executed

next line to execute





https://goo.gl/bwpHPo

The Stack Trace | Understanding Errors

```python
def print_hello(var):
    print("Hello!")
    x = 7 / var
    return x

def some_function(var):
    print("I am the function lord.")
    print(1 + 7 / 3)
    y = print_hello(var)
    print(y)

    return y
```

```python
some_function(0)
```

```
I am the function lord.
3.3333333333333335
Hello!

---------------------------------------------------------------
ZeroDivisionError                         Traceback (most r
<ipython-input-11-febbbdbb6e39> in <module>()
----> 1 some_function(0)

<ipython-input-10-3e7fc44e144f> in some_function(var)
      7     print("I am the function lord.")
      8     print(1 + 7 / 3)
----> 9     y = print_hello(var)
     10     print(y)
     11

<ipython-input-10-3e7fc44e144f> in print_hello(var)
      1 def print_hello(var):
      2     print("Hello!")
----> 3     x = 7 / var
      4     return x
      5

ZeroDivisionError: division by zero
```