CS 152: Functions

CS 152: Python for STEM

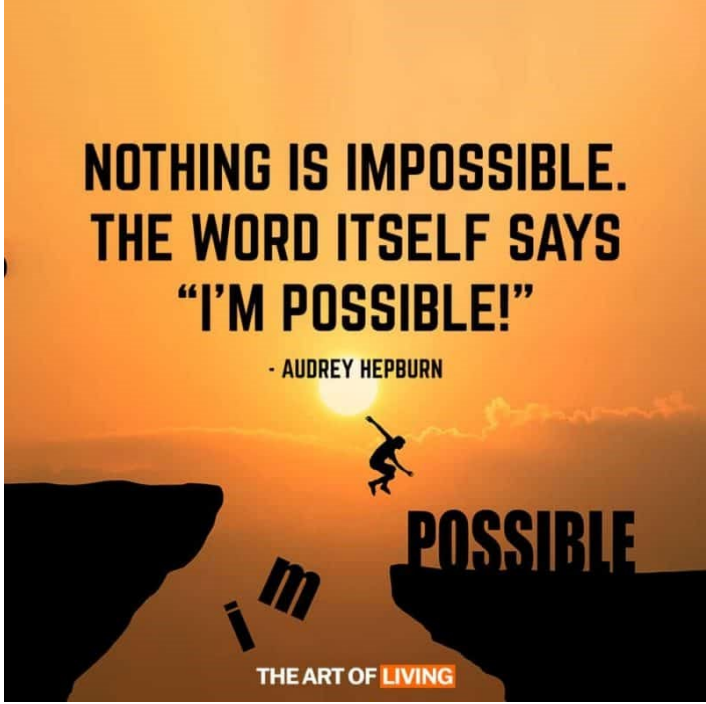
Colorado State University
Computer Science Department
Slides Originally Created by Albert Lionelle and Updated by Marcia Moraes

Colorado State University

Weekly Announcements!

TODO Reminders:

- Reading 3 (zyBooks) – you already should have done that for today's class ☺
- Lab 01 – Warm Up
- Reading 4 (zyBooks)
- Lab 02 - Application
- Reading 5 (zyBooks)



Recall Activity

- Individually
  - Access our Attendance assignment for today's class and write at least three concepts that you can remember from our last class
- With your neighbor(s)
  - Discuss what each other could remember. Did you remember the same things? What did you learn from each other?

Remembering - Peer Coding from Last Class

- Dr. Green is looking for a bank that will give the most return on her money over the next 5 years. She has P100,000.00 into a savings account. The standard equation to calculate principal plus interest at the end of a period is:
  - amount = P * (1 + I/M) ^ (N * M)
- Where:
  - P – principal (amount of money to invest)
  - I – interest (percentage rate the bank pays to the investor)
  - N – number of years (time for which the principal is invested)
  - M – compound interval (the number of times per year the interest is calculated and added to the principal)
- Think about what problem do you need to solve, how you are doing to solve it (write in English the steps to do that), write a Python code to solve that.

One Possible Solution

```
p = 10000000
n = 5
print("Enter the interest rate: ", end="")
i = float(input())
print("Enter the compound interval: ", end="")
m = float(input())
amount = p * (1 + i / m) ** (n * m)
print(f'Principal plus interest after {n:d} is {amount:.2f}')
```

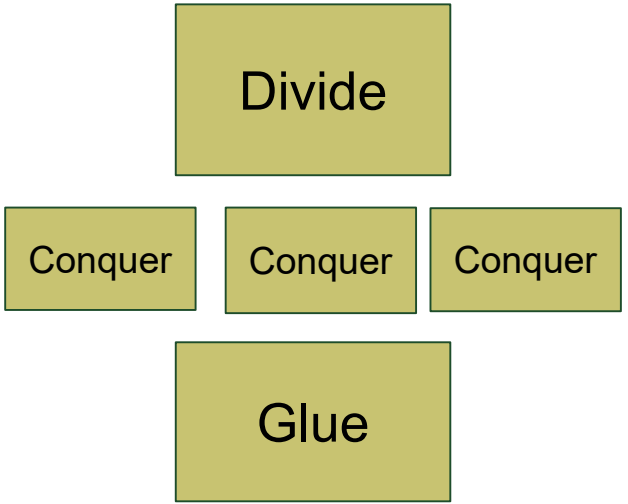
- What happen if we want to repeat this code to calculate the principal plus interest for another bank? Or for 10, 100 banks?

Code should be **Reusable**
Code should be **DRY**
- **D**on't **R**epeat **Y**ourself

Functions

Reusable Code

- Programming == Problem Solving
  - You look at the problem to solve
    - Clarify the problem and constraints
  - Break it up into *smaller* parts (Divide)
  - Outline the steps needed
    - Solve each step (Conquer)
  - Reassemble the pieces (Glue)
  - Completed program



Function

```
p = 10000000
n = 5
print("Enter the interest rate: ", end="")
i = float(input())
print("Enter the compound interval: ", end="")
m = float(input())
amount = p * (1 + i / m) ** (n * m)
print(f'Principal plus interest after {n:d} is {amount:.2f}')
```

- def
  - defines the start of a function
  - indents keep the 'code' with the function
  - spacing matters!
- parameters
  - allows for variables to the functions
  - print(your value)
    - function name print
    - your value is a parameter!

What should be the parameters for our code? Why?

Which commands are part of the function? Why?

Function

```
p = 10000000
n = 5
print("Enter the interest rate: ", end="")
i = float(input())
print("Enter the compound interval: ", end="")
m = float(input())
amount = p * (1 + i / m) ** (n * m)
print(f'Principal plus interest after {n:d} is {amount:.2f}')
```

```
def principal_plus_interest(i, m):
    amount = p * (1 + i / m) ** (n * m)
    print(f'Principal plus interest after {n:d} is {amount:.2f}')
```

```
p = 10000000
n = 5
print("Enter the interest rate: ", end="")
i = float(input())
print("Enter the compound interval: ", end="")
m = float(input())
principal_plus_interest(i, m)
```

How are we going to call the function we just created?

Could we improve this code a little bit more? How about p and n variables?

Return Values

- Better yet
- Functions do some work
  - and then return the answer

- Other programs can then use those answers
  - As they need / best for their problem
  - Always the best paradigm to follow
  - Notice **return** in get_real_code
    - returns the value, done with the function
  - Pure Functional - Most all functions should return something!

```
def get_real_code(code):
    return (code * 3 / 2) - 2.1

def print_machine_info(computer, code):
    print("The code to the " + computer + " is ", end='')
    print(get_real_code(code))

def use_formula(code):
    solve_cipher(get_real_code(code) * 10)
```

Your turn – one person coding - type the code above, and then as a group figure out ways to modify it! How do you test those changes?

Student Challenge

- Team Coding
- As group, write two functions
  - The first function takes in two parameters – first, last
    - It prints the ”welcome to the class (last), (first)”
  - The second function
    - Calls input to ask the client their first name
    - It calls input a second time to ask them their last name
    - it calls your first function to print out the result

- Have one person on the table code using their laptop and zybooks IDE
  - The person who codes has to limit their input. They follow the instructions of others
  - We will alternate who codes each class, so you won’t always be typing.

Recall Activity – From Your Readings

- What is incremental development?
- What is function stubs?
- What does the pass keyword mean? When is it used?
- Discuss your answer with your group

```
def steps_to_feet(num_steps):
    feet_per_step = 3
    feet = num_steps * feet_per_step
    return feet

def steps_to_calories(num_steps):
    pass

steps = int(input('Enter number of steps walked: '))

feet = steps_to_feet(steps)
print('Feet:', feet)

calories = steps_to_calories(steps)
print('Calories:', calories)
```

Recall Activity – From Your Readings

- What is a docstring?
- What is a help function? How would you use it in the example below?

```
def num_seats(airliner_type):
    """Determines number of seats on a plane"""
    #Function body statements ...

def ticket_price(origin, destination, coach=True, first_class=False):
    """Calculates the price of a ticket between two airports.
    Only one of coach or first_class must be True.

    Arguments:
    origin -- string representing code of origin airport
    destination -- string representing code of destination airport

    Optional keyword arguments:
    coach -- Boolean. True if ticket cost priced for a coach class ticket (default True)
    first_class -- Boolean. True if ticket cost priced for a first class ticket (default False)

    """
    #Function body statements ...
```