Week2: Functions (Ch. 5)

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY

This Week: Functions

Gentle Intro

Function Basics, Apply to Turtle Graphics

Main Function, ASCII Art Exercise

Variable Scope and Lifetime, Parameters

Intro to Parameters, Practice

Roadmap

☞ Functions

More Functions

Selection

Iteration

Midterm Exam

Strings

File I/O, Exceptions

Lists

List Algorithms

Dictionaries & Sets

Final Exam

3

A Gentle Intro

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY

4

The Concept

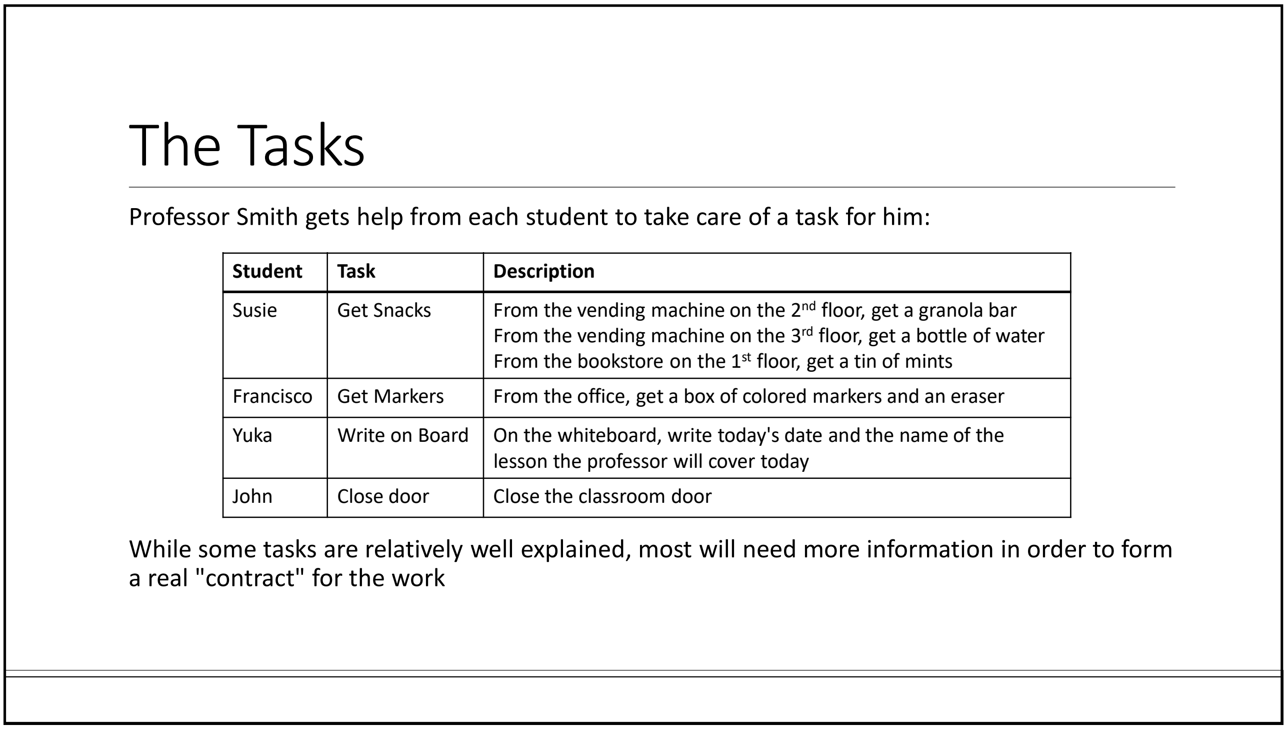
5

The Lazy Professor

Professor Smith has gotten lazy and has decided to outsource as much of his pre-lecture work as he can; this will make his life simpler but still achieve the same result

Luckily, he has eager and helpful students in his class, including Susie, Francisco, and Yuka, and John

6



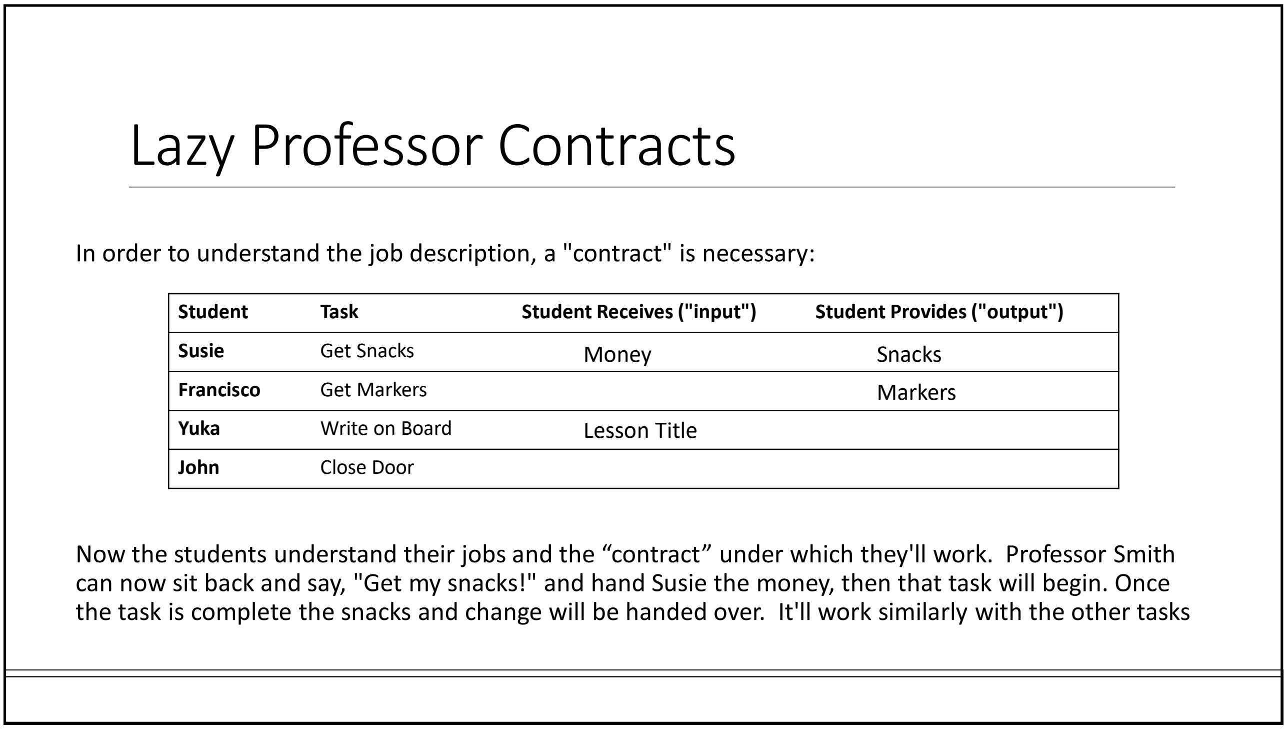
The Tasks

Professor Smith gets help from each student to take care of a task for him:

Student	Task	Description
Susie	Get Snacks	From the vending machine on the 2nd floor, get a granola bar From the vending machine on the 3rd floor, get a bottle of water From the bookstore on the 1st floor, get a tin of mints
Francisco	Get Markers	From the office, get a box of colored markers and an eraser
Yuka	Write on Board	On the whiteboard, write today's date and the name of the lesson the professor will cover today
John	Close door	Close the classroom door

While some tasks are relatively well explained, most will need more information in order to form a real "contract" for the work

7



Lazy Professor Contracts

In order to understand the job description, a "contract" is necessary:

Student	Task	Student Receives ("input")	Student Provides ("output")
Susie	Get Snacks	Money	Snacks
Francisco	Get Markers		Markers
Yuka	Write on Board	Lesson Title	
John	Close Door		

Now the students understand their jobs and the “contract” under which they'll work. Professor Smith can now sit back and say, "Get my snacks!" and hand Susie the money, then that task will begin. Once the task is complete the snacks and change will be handed over. It'll work similarly with the other tasks

8

Notes about the Contracts #1

From Susie's perspective (Get My Snacks), what comes “in” at the start of the task is Money; what goes “out” at the end are the snacks.

While Yuka (Write on the Board) does real work, nothing specific is handed over at the end of the task. We shouldn't confuse “handed over stuff” with valid and important work.

Each student will fulfill their contract with the professor. But exactly *how* they do it isn't specified and isn't of the professor’s concern. Susie might take the elevator or the stairs. She might get the mints before the water or vice-versa. As long as the task is completed and the contract is fulfilled, it’s nobody else's business. This makes each student’s work independent.

9

Notes about the Contracts #2

A student might decide to get helpers to do work (“subcontract” the work)

Example: Susie might have Dana get the granola bar; Lupe, the water; Chris, the mints

It’s of no consequence to the professor if this happens

The professor still gets what he wants; the contract is still fulfilled

The professor is unaware, in fact, of the change

Some contracts have “ins” and “outs;” some have only “ins;” some have only “outs.” Some have *neither*—they just do some work and don’t converse much. This is perfectly fine

10

Notes about the Contracts #3

Note also that each student doesn’t have to wonder or worry about who gets the results of their work; they are going to return the results to whoever initiated the job

If the professor taps you on the shoulder to start the work, the results go back to the professor

If Susie subcontracts the work (e.g., Susie taps Dana on the shoulder), then Dana’s results go to Susie

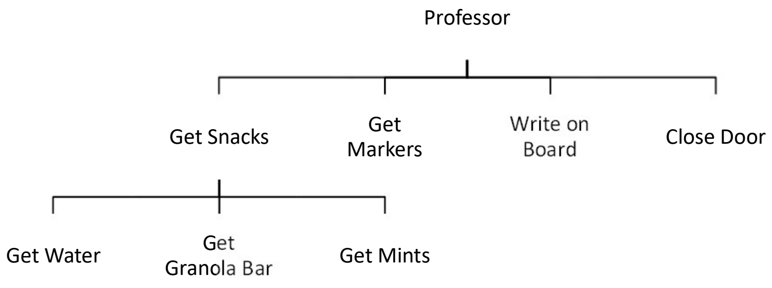
Wherever the job originates, that’s where the results go

This happens automatically in programs; the language knows where to direct the results without the individual worker having to think about it

11

Who Taps Whom: the Call Hierarchy

Here's what the work looks like when shown in a hierarchy. The professor initiates the work; students act. This call hierarchy assumes Susie is contracting out some of the snack work



Professor

Get Snacks

Get Markers

Write on Board

Close Door

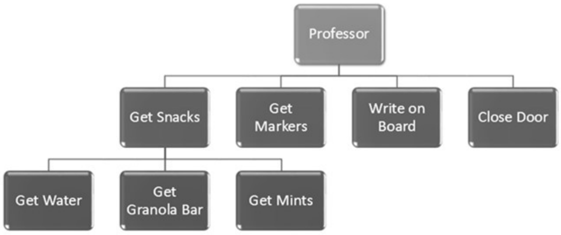
Get Water

Get Granola Bar

Get Mints

12

The Order of Things



The professor ("Main") calls Get Snacks

- Get Snacks calls Get Water
  - Get Water does its work, then returns control to Get Snacks, passing back the Water
- Get Snacks calls Get Granola Bar
  - Get Granola Bar does its work, then returns control to Get Snacks, passing back the Bar
- Get Snacks calls Get Mints
  - Get Mints does its work, then returns control to Get Snacks, passing back the Mints
- Get Snacks returns control to Main, passing back the snacks and the change

Main calls Get Markers

- Get Markers does its work, then returns control to main, passing back the markers and the eraser

Main calls Write on White Board

- Write on White Board does its work, then returns control to main

Let's interrupt a task and view its "call stack"

13

How many things at once?

In the analogy, we imagine several students scurrying around at once, but…

Q: How many things can your computer do at once?

A: Fewer than you think; much of it is smoke and mirrors. Multi-tasking gives each program a little slice of time

Even on modern computers with multiple processors, your program will run on a *single processor*; that CPU does *one thing at a time*. Fancy programs can work around this, but it's challenging to do

When one function calls another, the caller is "frozen" and the called function does its work. When the called function finishes, it goes away, the caller "unfrozen" to continue its work

This means that when we're deep in the hierarchy, there are potentially many "frozen" functions above us

14

Tasks and Contracts as Functions

To move this into the programming realm, each of the tasks become functions. Each of the requests Professor Smith makes is a “call” to a function. Here's what the function declarations look like, *conceptually*:

```
define GetSnacks(in:  money,   out: snacks, change)
        Go to the third floor, …

define GetMarkers(in:  nothing, out: markers, eraser)
        Go to the office, …

define WriteOnBoard(in: lesson title, out: nothing)
        Walk to the white board, …

define CloseDoor(in: nothing, out: nothing)
        Go to the door, close it, …
```

This is not Python yet; we’re on our way, though…

Student	Task	Prof Provides	Student Provides
Susie	Get Snacks	Money	Snacks, change
Francisco	Get Markers		Markers & eraser
Yuka	Write on Board	Lesson title	
John	Close Door		

Function Basics & Use Functions in Turtle

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY

What We’ll Cover

The chapter contains much more, but here we’ll touch on…

Function basics and benefits

Designing programs to use functions

Writing functions with parameters and return values

Using built-in modules

Creating your own module

17

Function Basics & Benefits

A function is a block of code that is **separate, independent,** and **named**

Python uses **indentation** to show what code is part of the function

Code in the function runs when you **mention the name** of the function

Functions **simplify** the main program (which often consists *only* of function calls)

You can call the same code over and over **without copying** the code

Functions allow for **focused testing**

- You can even write code to test the code, a programmatic “unit test”

Functions help with **top-down design** (breaking big problem down into smaller pieces)

Functions facilitate **teamwork** (assign out work by function)

18

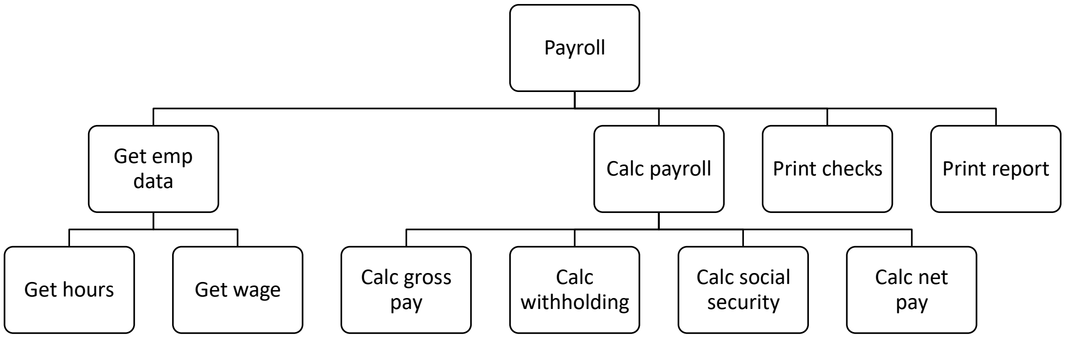
Designing Programs to Use Functions: Procedural Decomposition

We're faced with solving a big problem, doing payroll for an entire company. Let's use Top-down Design, creating a hierarchy chart as we go. Note the verb-oriented names.

Payroll

Designing Pgms to Use Functions

We're faced with solving a big problem, doing payroll for an entire company. Let's use Top-down Design, creating a hierarchy chart as we go. Note the verb-oriented names.



Payroll

Get emp data

Calc payroll

Print checks

Print report

Get hours

Get wage

Calc gross pay

Calc withholding

Calc social security

Calc net pay

Python Function Naming

Rules

Function names are a lot like variable names; the same rules apply

- You can’t use a Python keyword
- You can’t use spaces
- You must start with a letter, after that can come letters, numbers, or underscores

Conventions

- By convention, we use camelCasing
- By convention, function names start with verbs

21

Writing Functions in Python

Python doesn’t use *curly braces* to mark the starts and ends of things, like Java, C, JavaScript

It also doesn’t use *words* to mark them, like BASIC

Instead, Python uses **indentation** to show what belongs in a “block”:

```
def myFunction():
    print('This line is part of my function')
    print('So is this line')
print('But this line is not')
```

By convention, **indent four spaces** (one tab); this is a typical convention across languages

The def statement *defines* a function, but does not actually *run* its code; it’s waiting for us to call it later

22

Pause & Play

LET'S USE FUNCTIONS IN TURTLE WORLD

23

Let's Write Turtle Code Together

Task:

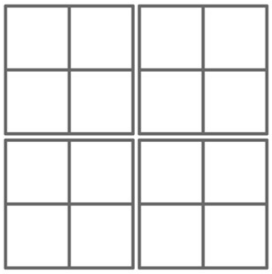
- Write a function that **draws a square** (100 units per side) in color dimgray
- Write a function that leverages that to **draw a window** with four square panes
- Write a function that leverages that to **draw four windows**, each 5 units away from the center

Remember to:

- Code incrementally; write a small amount of code at a time
- Test as you go; there's no pointing building on a shaky foundation

Question:

- What if we wanted to fill the panes with lightskyblue color?
- If we've done our work correctly, we should know *exactly* which function needs to be updated and shouldn't need to change code throughout



24

Turtle Window Code

```
def drawSquare():
    forward(100)
    right(90)
    forward(100)
    right(90)
    forward(100)
    right(90)
    forward(100)
    right(90)

def drawWindow():
    drawSquare()
    left(90)
    drawSquare()
    left(90)
    drawSquare()
    left(90)
    drawSquare()
    left(90)
```

```
def drawAllWindows():
    up()
    goto(-105, 105)
    down()
    drawWindow()
    up()
    goto(105, 105)
    down()
    drawWindow()
    up()
    goto(-105, -105)
    down()
    drawWindow()
    up()
    goto(105, -105)
    down()
    drawWindow()
```

```
from turtle import *
color('dimgray')
pensize(5)
drawAllWindows()
hideturtle()
done()
```

25

Main Function & Drawing ASCII Art

CSC110: INTRO TO COMPUTER PROGRAMMING WITH PYTHON

INSTRUCTOR: BILL BARRY

26

Writing a Main Function

By convention, we write one function called `main` whose job it is to drive the rest of the program

Often `main` is simple, only calling other functions

The *only* line that lives outside of any function is a single line that calls main

```
def sayHello():
    print("Hello")

def sayGoodbye():
    print("Goodbye")

def main():
    sayHello
    sayGoodbye

main()
```

Make sure you understand the exact flow of control here; let’s talk it through

27

Pause & Play

LET’S USE FUNCTIONAL DECOMPOSITION AND WRITE SIMPLE FUNCTIONS

28

Let’s Write Code Together

Let’s write a program that creates the output shown at right

```
+------+
|      |
|      |
+------+

+------+
|      |
|      |
+------+
```

Version #1

Write 9 lines of code, cutting and pasting a bit.

(Surely there’s a better way!)

Version #2

Create a function called drawBox, then write a main program that calls it *twice*

Version #n

Any other suggestions that could reduce redundancy?

29

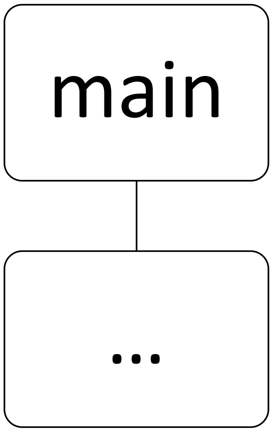
Draw Boxes Code

```
def drawBox():
    print('+------+')
    print('|      |')
    print('|      |')
    print('+------+')

def main():
    drawBox()
    drawBox()

main()
```

30

Challenge #1: Draw a hierarchy chart, then write code to produce this output

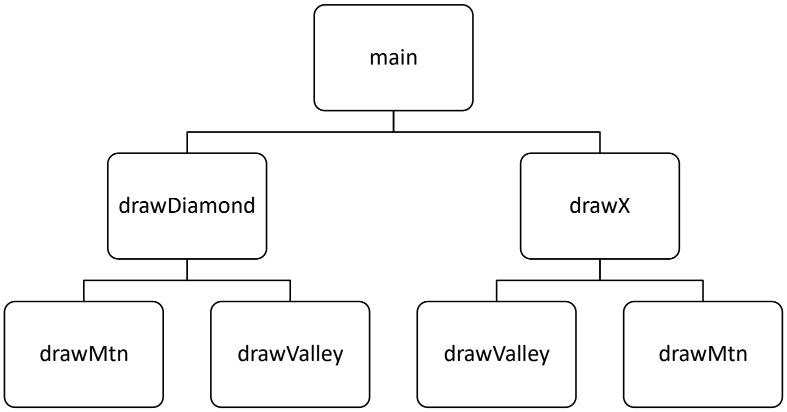
```
  /\
 /  \
/    \
\    /
 \  /
  \/

\    /
 \  /
  \/
  /\
 /  \
/    \
```



31

Challenge #1: Draw a hierarchy chart, then write code to produce this output

```
  /\
 /  \
/    \
\    /
 \  /
  \/

\    /
 \  /
  \/
  /\
 /  \
/    \
```


main

drawDiamond

drawX

drawMtn

drawValley

drawValley

drawMtn

32

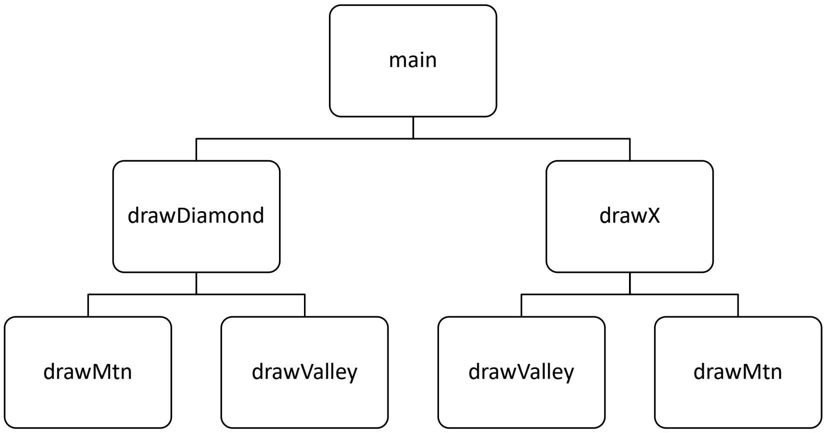
Challenge #1 Code

```
def drawMtn():
    print('   /\\')
    print('  /  \\')
    print(' /    \\')

def drawValley():
    print(' \\    /')
    print('  \\  /')
    print('   \\/')

def drawDiamond():
    drawMtn()
    drawValley()
    print()
```

```
def drawX():
    drawValley()
    drawMtn()
    print()

def main():
    drawDiamond()
    drawX()

main()
```

Challenge #2: Modify your design and code so it also produces this output

```
   /\
  /  \
 /    \
+------+
|      |
|      |
+------+
| Mars |
|Bound!|
+------+
|      |
|      |
+------+
   /\
  /  \
 /    \
```


main

drawDiamond

drawX

drawMtn

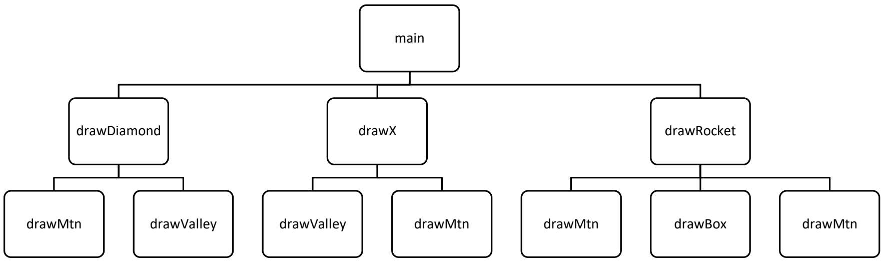
drawValley

drawValley

drawMtn

Challenge #2: Modify your design and code so it also produces this output

```
   /\
  /  \
 /    \
+------+
|      |
|      |
+------+
| Mars |
|Bound!|
+------+
|      |
|      |
+------+
   /\
  /  \
 /    \
```



35

Challenge #2 Code

```
def drawBox():
    print('+------+')
    print('|      |')
    print('|      |')
    print('+------+')

def drawMtn():
    print('   /\\')
    print('  /  \\')
    print(' /    \\')

def drawValley():
    print(' \\    /')
    print('  \\  /')
    print('   \\/')

def drawDiamond():
    drawMtn()
    drawValley()
    print()
```

```
def drawX():
    drawValley()
    drawMtn()
    print()

def drawRocket():
    drawMtn()
    drawBox()
    print('| Mars |')
    print('|Bound!|')
    drawBox()
    drawMtn()
    print

def main():
    drawDiamond()
    drawX()
    drawRocket()

main()
```

36

Variable Scope and Lifetime, Function Parameters

37

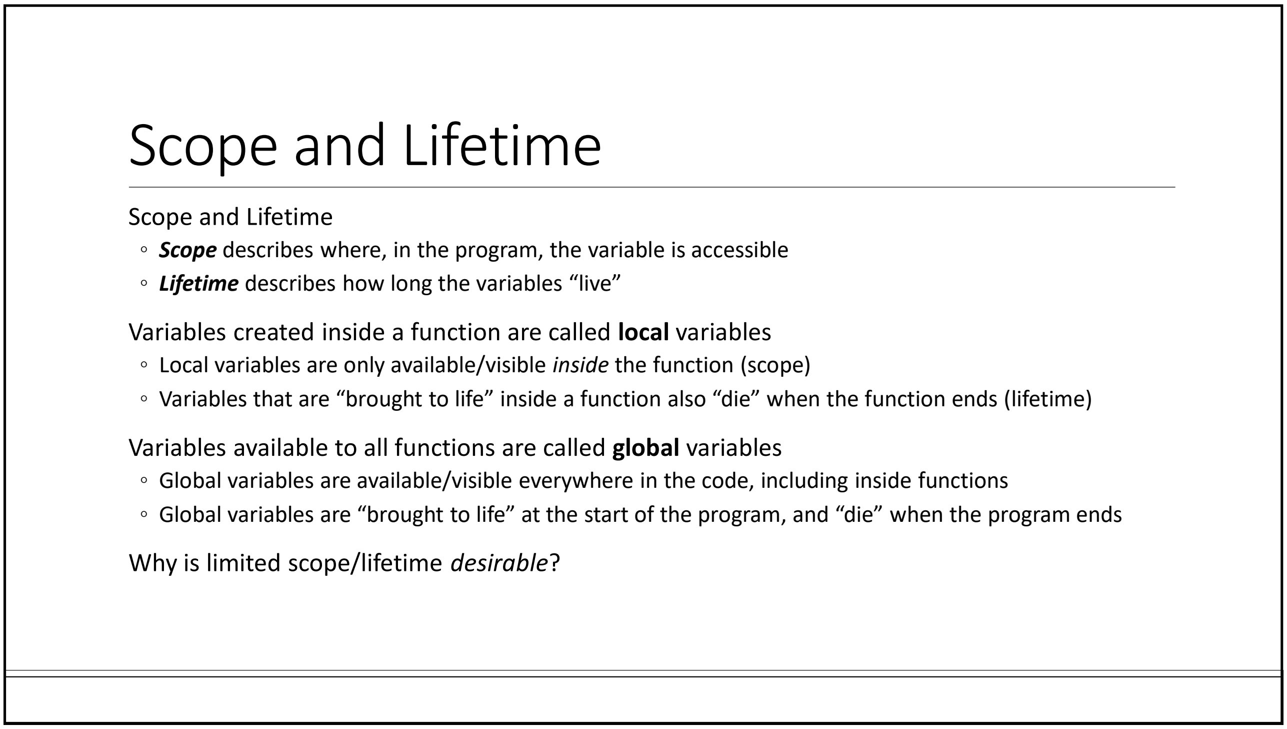
Scope and Lifetime Analogies

There are two terms we'll introduce today: **global** and **local**

Global variables are things that need to last a long time, like a **class notebook** you use for the whole quarter. You write in it, take it home, make sure you know where it is, and bring it back for each class or study session

Local variables are more temporary, something that's used for a specific amount of time, then discarded. These are like **exam scratch paper**. You scribble and draw on it, but when the exam is over, you crumple it up and throw it in the recycle bin

38



Scope and Lifetime

Scope and Lifetime

- ***Scope*** describes where, in the program, the variable is accessible
- ***Lifetime*** describes how long the variables "live"

Variables created inside a function are called **local** variables

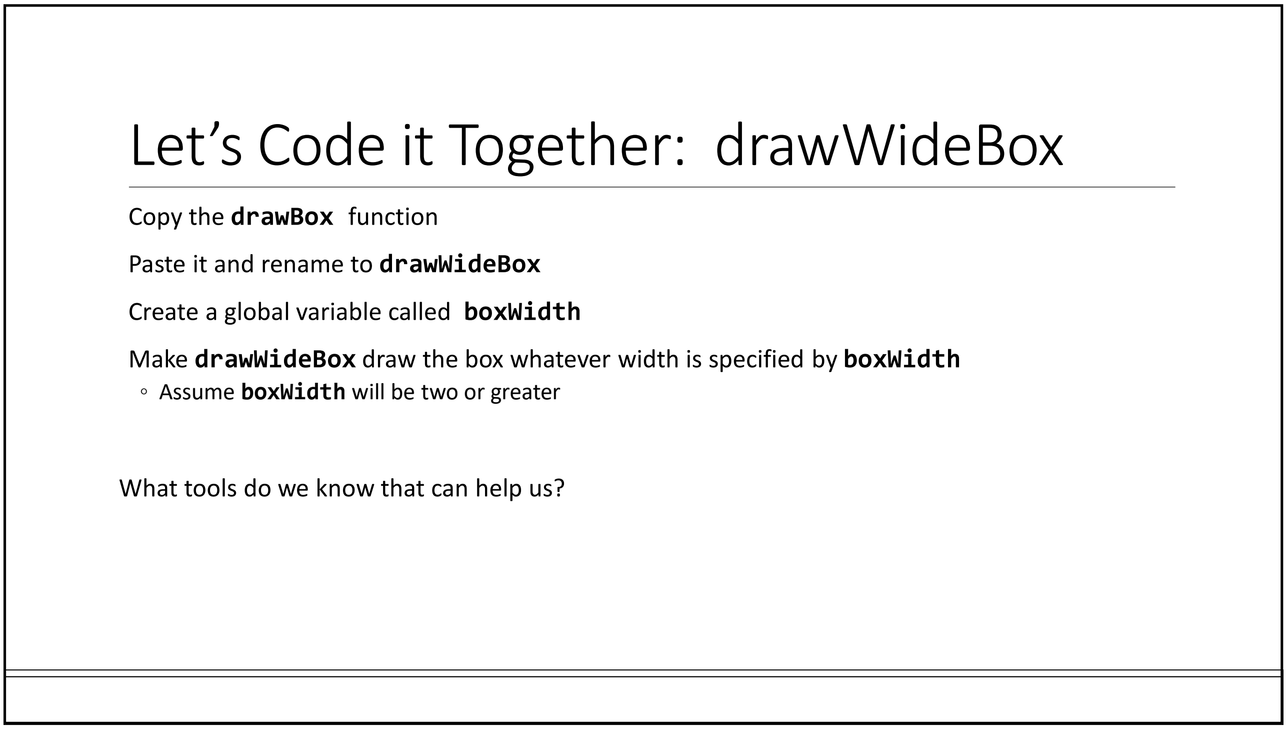
- Local variables are only available/visible *inside* the function (scope)
- Variables that are "brought to life" inside a function also "die" when the function ends (lifetime)

Variables available to all functions are called **global** variables

- Global variables are available/visible everywhere in the code, including inside functions
- Global variables are "brought to life" at the start of the program, and "die" when the program ends

Why is limited scope/lifetime *desirable*?

39



Let's Code it Together: drawWideBox

Copy the **`drawBox`** function

Paste it and rename to **`drawWideBox`**

Create a global variable called **`boxWidth`**

Make **`drawWideBox`** draw the box whatever width is specified by **`boxWidth`**

- Assume **`boxWidth`** will be two or greater

What tools do we know that can help us?

40

drawWideBox

```
BOX_WIDTH = 10    # declared outside any function, a global variable

def drawWideBox():
    print('+', '-' * (BOX_WIDTH - 2), '+', sep='')
    print('|', ' ' * (BOX_WIDTH - 2), '|', sep='')
    print('|', ' ' * (BOX_WIDTH - 2), '|', sep='')
    print('+', '-' * (BOX_WIDTH - 2), '+', sep='')

def main():
    drawWideBox()

main()
```

41

Let’s Code it Together: Get Input

Add a function called **`getBoxWidth`**

In it, ask the user how wide they want their boxes to be, then store their answer in the global variable **`boxWidth`**

Modify **`main`** to call **`getBoxWidth`** before **`drawWideBox`**

What new challenges do we encounter?

42

drawWideBox with UI

```
boxWidth = 3

def getBoxWidth():
    global boxWidth    # global keeps Python from creating a new local variable
    boxWidth = int(input('How wide do you want your boxes? '))

def drawWideBox():
    print('+', '-' * (boxWidth - 2), '+', sep='')   # no global keyword needed here
    print('|', ' ' * (boxWidth - 2), '|', sep='')
    print('|', ' ' * (boxWidth - 2), '|', sep='')
    print('+', '-' * (boxWidth - 2), '+', sep='')

def main():
    getBoxWidth()
    drawWideBox()

main()
```

While this is an interesting exercise, in general we should avoid the use of globals

43

Parameters

USING THE FUNCTION’S “INBOX” TO PASS DATA

44

A Quick Intro to Parameters

In our Lazy Professor analogy, we talked about each task (function) having an Inbox and an Outbox

The way languages implement the “inbox” is via *parameters*

A parameter is a variable listed inside the parentheses that follow the function name in a function definition

When the function is called, the caller *passes* data (*argument*). The function *receives* the data and stores it in the parameter variable

The function can then use that variable in its code

45

Let’s Code it Together: Parameters

Start with the code at right

Get rid of the global variable

Modify **`drawWideBox`** to accept a single *parameter*, the width of the box; use that data to draw a box of the desired width

In main, ask the user for the desired width and pass it as an argument in a call to drawWideBox

Save and test your code

```
BOX_WIDTH = 10

def drawWideBox():
    print("+", "-" * (BOX_WIDTH - 2), "+", sep = "")
    print("|", " " * (BOX_WIDTH - 2), "|", sep = "")
    print("|", " " * (BOX_WIDTH - 2), "|", sep = "")
    print("+", "-" * (BOX_WIDTH - 2), "+", sep = "")

def main():
    drawWideBox()

main()
```

46

drawWideBox with a Parameter

```
def drawWideBox(boxWidth):
    print('+', '-' * (boxWidth - 2), '+', sep='')
    print('|', ' ' * (boxWidth - 2), '|', sep='')
    print('|', ' ' * (boxWidth - 2), '|', sep='')
    print('+', '-' * (boxWidth - 2), '+', sep='')

def main():
    desiredWidth = int(input('How wide do you want your boxes? '))
    drawWideBox(desiredWidth)

main()
```

Analogy: on the *calling* side, the value of the *argument* is written onto a piece of scratch paper. On the *function* side, a local variable is created, its value assigned to the *parameter's* value (what's on the scratch paper)

47

Pause & Practice

LET'S PRACTICE SIMPLE FUNCTIONS THAT TAKE PARAMETERS

48

Function Challenge

In a new file saved as **FunctionPractice.py**, write a function called **`showRightTriangleArea`**. It should accept two **parameters**, base and the height, and calculate area using this formula: area = 1/2 x base x height. It should then display nicely labeled output as shown in this sample run, with two digits after the decimal in the results:

```
Enter right triangle base: 3
Enter right triangle height: 4

The area of a right triangle
with base of 3 units
and height of 4 units
is 6.00 square units
```

Write a **`main`** function that asks the user for the base and the height, then calls **`showRightTriangleArea`** and passes the two arguments, in the expected order

Write one line of code that calls the main function. Run and test your work

49

Function Challenge Solution

```
def showRightTriangleArea(base, height):
    area = 0.5 * base * height
    print()
    print("The area of a right triangle")
    print("with base of", base, "units")
    print("and height of", height, "units")
    print("is", format(area, ".2f"), "square units")

def main():
    triBase = int(input("Enter right triangle base: "))
    triHeight = int(input("Enter right triangle height: "))
    showRightTriangleArea(triBase, triHeight)

main()
```

50

The End

51