Objectives

- Defining your own functions
  - Control flow
  - Scope, variable lifetime

1

Review

- How do we add animation to our graphics programs?
- What are benefits of functions?

2

Looking behind the curtain…

DEFINING OUR OWN FUNCTIONS

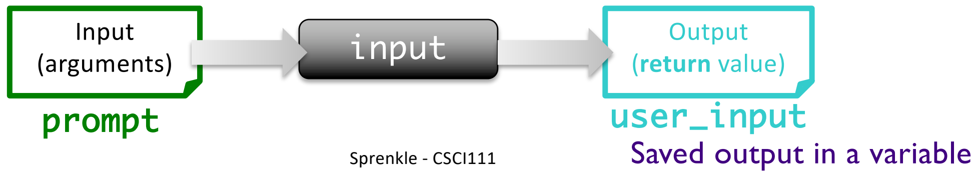
Functions

- We've used functions
  - Built-in functions: `input`, `eval`
  - Functions from modules, e.g., `math` and `random`
- Benefits
  - Reuse, reduce code
  - Easier to read, write (because of ***abstraction***)

Today, we'll learn how to
define our own functions!

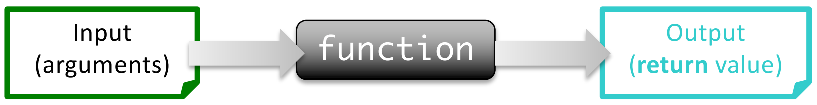
Review: Functions

- Function is a **black box**
  - Implementation doesn't matter
  - Only care that function generates appropriate output, given appropriate input
- Example:
  - Didn't care how `input` function was implemented
  - Use: `user_input = input(prompt)`



Creating Functions

- A function can have
  - 0 or more inputs
  - 0 or 1 outputs
- When we define a function, we know its inputs and if it has output



Writing a Function

- Goal: a function that moves a circle to a new location
- Recall:

```
# create the circle in the center of the window and draw it
midPoint = Point(canvas.getWidth()/2, canvas.getHeight()/2)
myCircle = Circle(midPoint, CIRCLE_RADIUS)
myCircle.draw(canvas)

# get where the user clicked
new_point = canvas.getMouse()

# Move the circle to where the user clicks
centerPoint = myCircle.getCenter()

dx = new_point.getX() - centerPoint.getX()
dy = new_point.getY() - centerPoint.getY()

myCircle.move(dx,dy)

canvas.getMouse()
```

A Function to Move a Circle

Inputs/Parameters:

The circle to move The point to move the circle to

```
def moveCircle( circle, newCenter ):
    """
    Move the given Circle circle to be centered
    at the Point newCenter
    """
    centerPoint = circle.getCenter()

    diffInX = newCenter.getX() - centerPoint.getX()
    diffInY = newCenter.getY() - centerPoint.getY()

    circle.move(diffInX, diffInY)
```

A Function to Move a Circle

Keyword *Function Name* *Input Name/ Parameter*

```
def moveCircle( circle, newCenter ):
    """
    Move the given Circle circle to be centered
    at the Point newCenter
    """
    centerPoint = circle.getCenter()

    diffInX = newCenter.getX() - centerPoint.getX()
    diffInY = newCenter.getY() - centerPoint.getY()

    circle.move(diffInX, diffInY)
```

Function header

Function documentation

Body (or function definition)

Defining a Function

- Gives a name to some code that you'd like to be able to call again
- Analogy:
  - **Defining a function**: saving name, phone number, etc. in your contacts
  - **Calling a function**: calling that number

Parameters

- The inputs to a function are called ***parameters*** or ***arguments***, depending on the context
- When ***calling***/using functions, arguments must appear in same order as in the function header
  - Example: `round(x, n)`
    - **x** is the `float` to round
    - **n** is `int` of decimal places to round **x** to

Parameters

- **Formal Parameters** are the variables named in the function definition
- **Actual Parameters** or **Arguments** are the variables or literals that really get used when the function is called.

Formal

Actual

Defined: `def round(x, n) :`

Use: `roundCelc = round(celcTemp, 3)`

Formal & actual parameters must match in ***order***, ***number***, and ***type***!

Calling the Function

```
# create the circle in the center of the window and draw it
midPoint = Point(canvas.getWidth()/2, canvas.getHeight()/2)
myCircle = Circle(midPoint, CIRCLE_RADIUS)
myCircle.draw(win)

# get where the user clicked
new_point = canvas.getMouse()

moveCircle( myCircle, new_point )
```

The circle to move   The point to move the circle to

Same as calling someone else's functions …

Compare the code…

`circleShiftWithFunction.py`

13

Writing a Function

- Let's look at one more example
- I want a function that averages two numbers

• What is the input to this function?
• What is the output from this function?

14

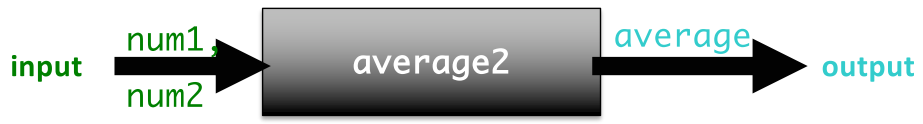
Writing a Function

- I want a function that averages two numbers
- What is the input to this function?
  - The two numbers
- What is the output from this function?
  - The average of those two numbers, as a float

These are key questions to ask yourself when designing your own functions.

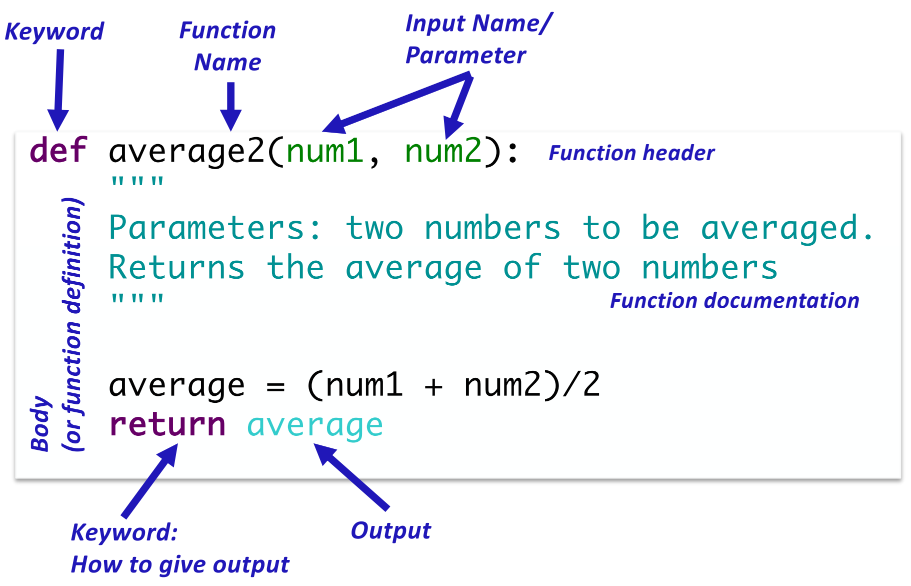
- Inputs: What are the parameters?
- Output: What is getting returned?

Averaging Two Numbers



- **Input**: the two numbers
- **Output**: the average of two numbers

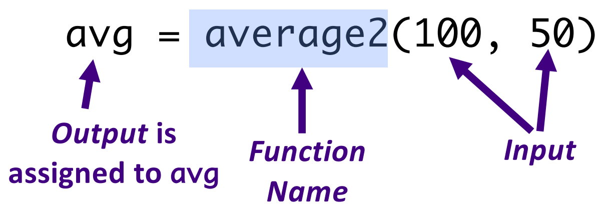
Syntax of Function Definition



```python
def average2(num1, num2):
    """
    Parameters: two numbers to be averaged.
    Returns the average of two numbers
    """

    average = (num1 + num2)/2
    return average
```

Calling your own functions

Same as calling someone else's functions …



```python
avg = average2(100, 50)
```

`average2.py`

Functions: Similarity to Math

- In math, a function definition looks like:

$$\boxed{f(x) = x^2 + 2}$$

- Plug values in for x
- Example:
  - f(3) = 3^2 + 2 = 11
  - 3 is your ***input***, assigned to x
  - 11 is output

Feb 2, 2022 Sprenkle - CSCI111 19

19

Function Output

- When the code reaches a statement like

```
return x
```

  - The function stops executing
  - `x` is the **output *returned*** to the place where the function was called
- For functions that don't have explicit output, `return` does not have a value with it, e.g.,

```
return
```

- Optional: don't *need* to have `return`
  - Function *automatically* returns at the end (like `moveCircle`)

Feb 2, 2022 Sprenkle - CSCI111 20

20

Flow of Control

- When program calls a function, the program jumps to the function and executes it
- After executing the function, the program returns to the same place in the *calling code* where it left off

Calling code:

```
# Make conversions
dist1 = 100
miles1 = metersToMiles(dist1)
```

Value of `dist1` (100) is assigned to `meters`

```
def metersToMiles(meters) :
    M2MI=.0006215
    miles = meters * M2MI
    return miles
```

average2.py

Function Definition Example without Output

Keyword ***Function Name*** ***Input Name/ Parameter***

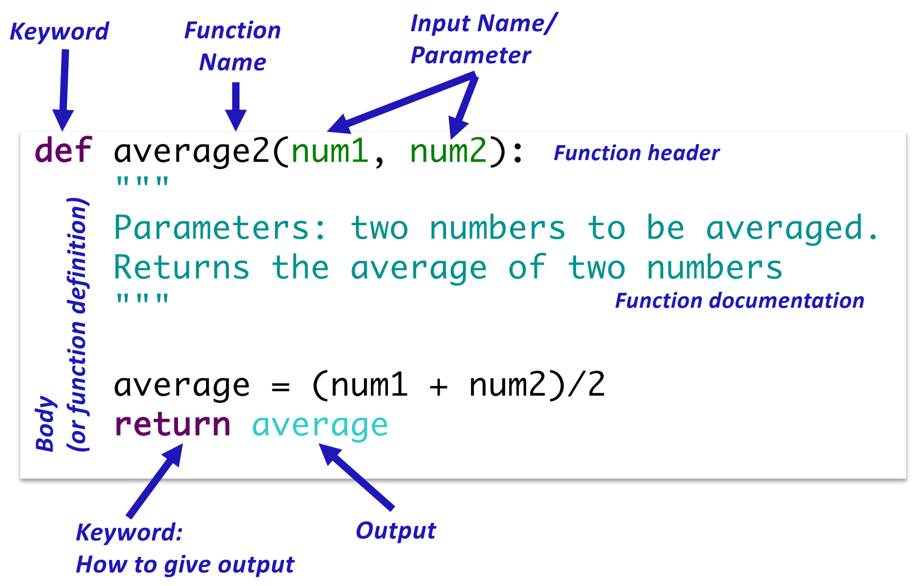
```
def moveCircle( circle, newCenter ):
    """
    Move the given Circle circle to be centered
    at the Point newCenter
    """
    centerPoint = circle.getCenter()

    diffInX = newCenter.getX() - centerPoint.getX()
    diffInY = newCenter.getY() - centerPoint.getY()

    circle.move(diffInX, diffInY)
```

Function header

Function documentation

Body (or function definition)

Function Definition with Output



```
def average2(num1, num2):
    """
    Parameters: two numbers to be averaged.
    Returns the average of two numbers
    """

    average = (num1 + num2)/2
    return average
```

Using `print` vs `return`

- `print` is for displaying information
- Don't always want to display the output of a function
- `return` gives us more flexibility about what we do with the output from a function
- Example:

```
avg = average2(num1, num2)
print("The average is", round(avg, 2) )
```

We don't want the "raw" value from *average2* displayed when the function is called. We want to process that value so that we only display it to two decimal places. Maybe another place we call it, we want to round the result to 4 decimal places.

`return` vs `print`

- In general, whenever we want output from a function, we'll use `return`
  - More flexible, reusable function
  - Let whoever called the function figure out what to display
- Use `print` for
  - Debugging your function (then remove before final submission)
    - Otherwise, unintended side effect of calling the function
  - When you have a function that is supposed to *display* something
    - Sometimes, displaying something is what you want.

Function Input and Output

- What does this function do?
- What is its input? What is its output?

Constants and comments are in example program

```
def printVerse(animal, sound):
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
    print("And a", sound, ",", sound, "there")
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, ",", sound)
    print(BEGIN_END + EIEIO)
    print()
```

What does this function do if called as `printVerse("pig", "oink")`? As `printVerse("oink", "pig")`?

Function Input and Output

- 2 inputs: `animal` and `sound`
- 0 outputs
  - ➢ ***Displays*** something but does not **`return`** anything (None)

```
def printVerse(animal, sound):
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
    print("And a", sound, ",", sound, "there")
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, ",", sound)
    print(BEGIN_END + EIEIO)
    print()
```

Function exits here

28

Words in Different Contexts

"Time flies like an arrow.
Fruit flies like bananas."
— Groucho Marx.

- **Output** from a *function*
  - ➢ What is ***returned*** from the function
  - ➢ If the function displays something, it's what the function ***displays*** or prints (rather than outputs).
- **Output** from a *program*
  - ➢ What is displayed by the program

29

PROGRAM ORGANIZATION

Where are Functions Defined?

- Functions can go inside program script
  - If no **`main`**() function, defined ***before*** use/called
    - `average2.py`
  - If `main()` function, defined anywhere in script

- Functions can go inside a separate ***module***

Program Organization: `main` function

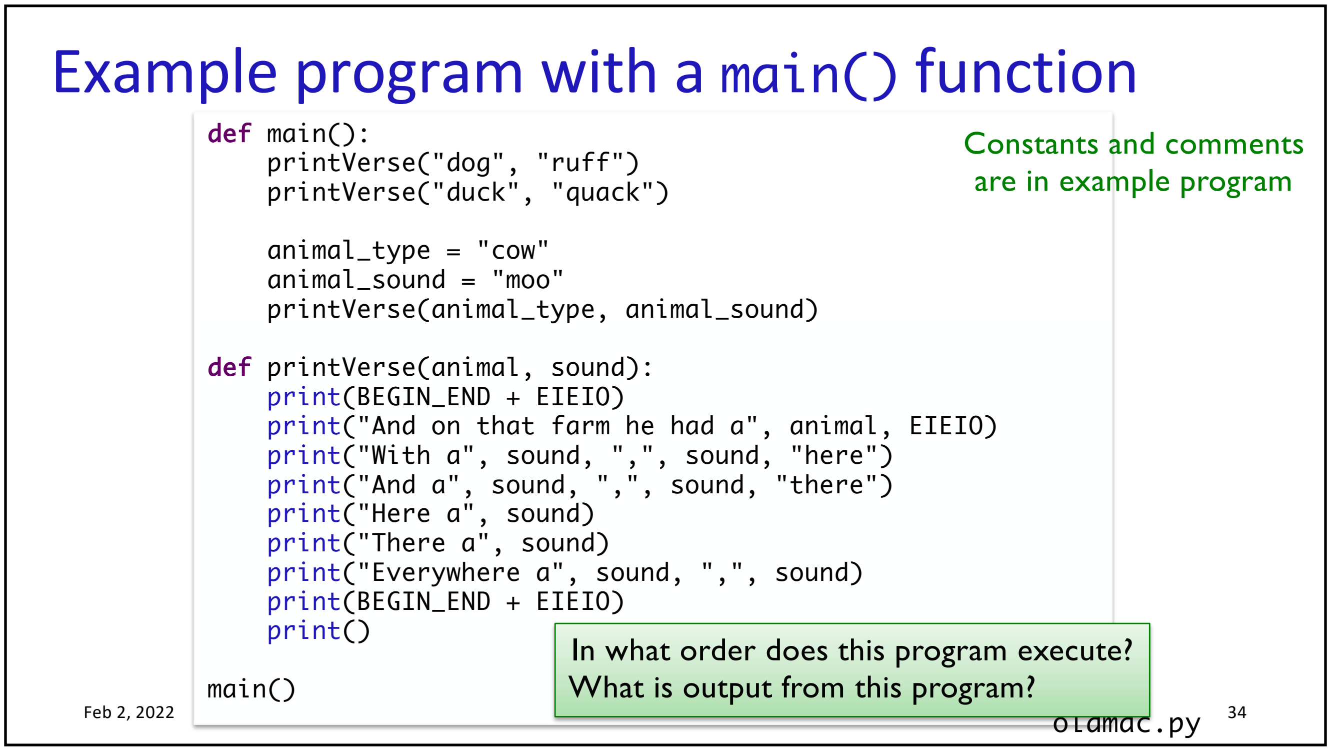
- In many languages, you put the "driver" for your program in a `main` function
  - You can (and should) do this in Python as well
- Typically `main` functions are defined near the top of your program
  - Readers can quickly see an overview of what program does
- `main` usually takes no arguments
  - Example: `def main():`

32

Using a `main` Function

- Call `main()` at the bottom of your program
- Side effects:
  - Do not need to define functions before `main` function
  - `main` can "see" all other functions
- `main` is a function that calls other functions
  - *Any* function can call other functions !

33



Example program with a `main()` function

```
def main():
    printVerse("dog", "ruff")
    printVerse("duck", "quack")

    animal_type = "cow"
    animal_sound = "moo"
    printVerse(animal_type, animal_sound)

def printVerse(animal, sound):
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
    print("And a", sound, ",", sound, "there")
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, ",", sound)
    print(BEGIN_END + EIEIO)
    print()

main()
```

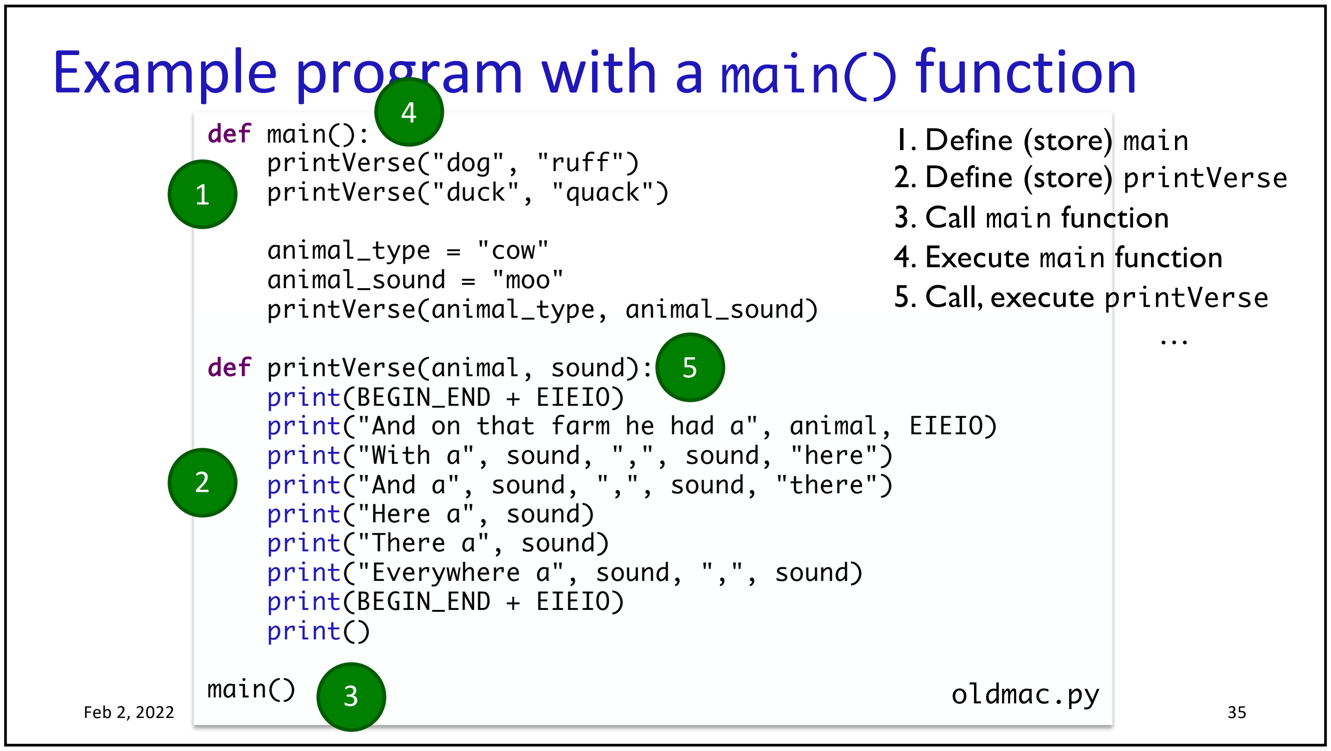
`oldmac.py`

Constants and comments are in example program

In what order does this program execute?
What is output from this program?

Feb 2, 2022 34

34



Example program with a `main()` function

```
def main():   4
    printVerse("dog", "ruff")
 1  printVerse("duck", "quack")

    animal_type = "cow"
    animal_sound = "moo"
    printVerse(animal_type, animal_sound)

def printVerse(animal, sound):   5
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a", animal, EIEIO)
    print("With a", sound, ",", sound, "here")
 2  print("And a", sound, ",", sound, "there")
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, ",", sound)
    print(BEGIN_END + EIEIO)
    print()

main()   3
```

`oldmac.py`

1. Define (store) `main`
2. Define (store) `printVerse`
3. Call `main` function
4. Execute `main` function
5. Call, execute `printVerse`

…

Feb 2, 2022 35

35

Summary: Program Organization

- Larger programs require functions to maintain readability
  - Use `main()` and other functions to break up program into smaller, more manageable chunks
  - "Abstract away" the details
- As before, can still write smaller scripts without any functions
  - Can try out functions using smaller scripts
- Need the `main()` function when using other functions to keep "driver" at top
  - Otherwise, functions need to be defined before use

36

Why Write Functions?

- Allows you to break up a problem into *smaller*, more *manageable* parts
- Makes your code easier to *understand*
- Hides implementation details (*abstraction*)
  - Provides *interface* (input, output)
- Makes part of the code *reusable* so that you:
  - Only have to write function code once
  - Can debug it all at once
    - Isolates errors
  - Can make changes in one function (*maintainability*)

Feb 2, 2022

Similar to benefits of OO Programming

37

37

VARIABLE LIFETIMES AND SCOPE

What does this program output?

```
def main():
  x = 10
  sum = sumEvens( x )
  print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit):
  total = 0
  for x in range(0, limit, 2):
        total += x
  return total

main()
```

mystery.py

Function Variables

```
def main():
   x = 10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit):
   total = 0
   for x in range(0, limit, 2):
         total += x
   return total

main()
```

Why can we name two different variables x?

mystery.py

Tracing through Execution

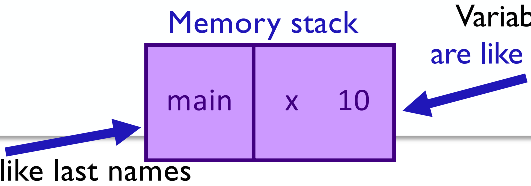
Defines functions

```
def main():
   x = 10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit):
   total = 0
   for x in range(0, limit, 2):
         total += x
   return total

main()
```

When you call main(), that means you want to execute this function

Function Variables

```
def main() :
   x=10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
   total = 0
   for x in range(0, limit, 2):
          total += x
   return total

main()
```



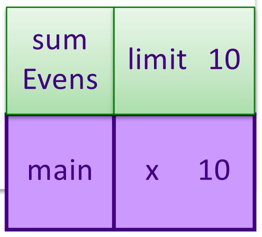
Variable names
are like first names

Function names are like last names
Define the ***SCOPE*** of the variable

Function Variables

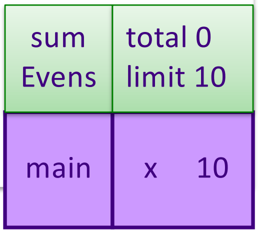
```
def main() :
   x=10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
   total = 0
   for x in range(0, limit, 2):
          total += x
   return total

main()
```

Called the function sumEvens
Add its parameters to the stack



Function Variables

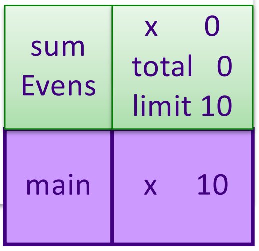
```
def main() :
   x=10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
   total = 0
   for x in range(0, limit, 2):
          total += x
   return total

main()
```



44

Function Variables

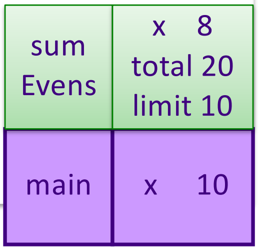
```
def main() :
   x=10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
   total = 0
   for x in range(0, limit, 2):
          total += x
   return total

main()
```



45

Function Variables

```
def main() :
   x=10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
   total = 0
   for x in range(0, limit, 2):
          total += x
   return total

main()
```

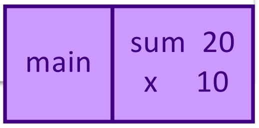


46

Function Variables

```
def main() :
   x=10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
   total = 0
   for x in range(0, limit, 2):
          total += x
   return total

main()
```

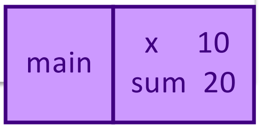
Function sumEvens returned
- no longer have to keep track of its variables on stack
- lifetime of those variables is over



47

Function Variables

```
def main() :
   x=10
   sum = sumEvens( x )
   print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
   total = 0
   for x in range(0, limit, 2):
          total += x
   return total

main()
```



Variable Scope

- Functions can have the same parameter and variable names as other functions
  - Need to look at the variable's ***scope*** to determine which one you're looking at
  - Use the ***stack*** to figure out which variable you're using
- Scope levels
  - **Local** scope (also called **function scope**)
    - Can only be seen within the function
  - **Global** scope (also called **file scope**)
    - Whole program can access
    - More on these later

Writing Documentation for Functions

- Good style: Each function* ***must*** have a comment that documents its use
- Describes functionality at a high-level
- Include the *precondition*, *postcondition*
- Describe the parameters (their types) and the result of calling the function (precondition and postcondition may cover this)
- The exact format matters less than that the content is there
  - I'll show a few different ways to write the documentation

Writing Comments for Functions

- Include the function's pre- and post- conditions
- **Precondition**: Things that must be true for function to work correctly
  - E.g., num must be even; circle must be a Circle object
- **Postcondition**: Things that will be true when function finishes (if precondition is true)
  - E.g., the returned value is the max; circle will be moved to the new point

Example Documentation

- Describes at high-level
- Describes parameters

```
def printVerse(animal, sound):
    """
    Prints a verse of Old MacDonald, plugging in the
    animal and sound parameters (which are strings),
    as appropriate.
    """
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a " + animal + EIEIO)
    …
```

Comment style: **Docstring**
"documentation string"

When you use the `help` function, it shows the docstrings.

Another Example Comment

- Describes at high-level
- Describes parameters

```
def average2(num1, num2):
    """
    Parameters: two numbers to be averaged
    Returns the average of the two numbers
    """
    average = (num1 + num2)/2
    return average
```

Comment style: **Docstring**
"documentation string"

When you use the `help` function, it shows the docstrings.

Write the Docstring Comment for sumEvens

```
def main() :
  x=10
  sum = sumEvens( x )
  print("The sum of even #s up to", x, "is", sum)

def sumEvens(limit) :
  """



  """
  total = 0
  for x in range(0, limit, 2):
        total += x
  return total

main()
```

Looking Ahead

- BI – Google Search
- Lab 3 due Friday
- Exam next Friday
  - Prep document up soon