

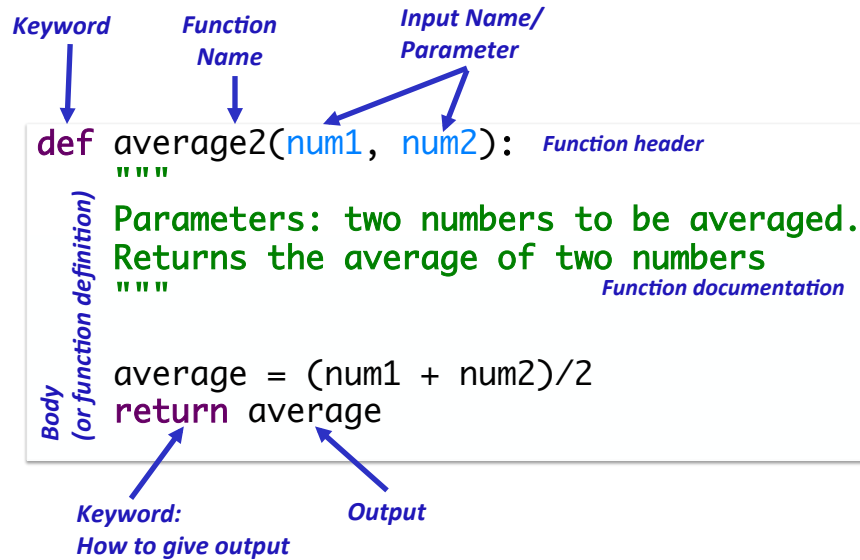
Objectives

- Defining your own functions
 - Control flow
 - Scope, variable lifetime
- Testing
- Refining our development process
- BI: Net Neutrality

Review

- What are benefits of functions?
- What is the syntax for creating a function?
- What is the special keyword that means “this is the output for the function”?

Review: Syntax of Function Definition



The diagram illustrates the syntax of a Python function definition with the following components and annotations:

- Keyword:** Points to the `def` keyword.
- Function Name:** Points to the function name `average2`.
- Input Name/Parameter:** Points to the parameters `num1` and `num2`.
- Function header:** Points to the entire line `def average2(num1, num2):`.
- Function documentation:** Points to the docstring `"""Parameters: two numbers to be averaged. Returns the average of two numbers"""`.
- Body (or function definition):** Points to the function body, which includes the calculation `average = (num1 + num2)/2` and the `return average` statement.
- Keyword: How to give output:** Points to the `return` keyword.
- Output:** Points to the variable `average` being returned.

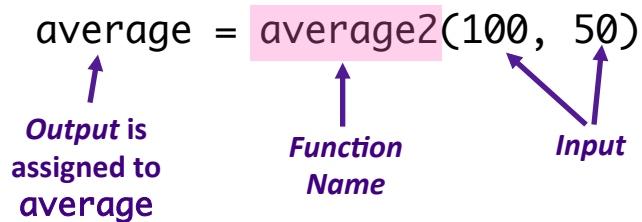
Feb 2, 2018

Sprenkle - CSCI111

3

Review: Calling your own functions

Same as calling someone else's functions ...



The diagram illustrates the syntax of a function call with the following components and annotations:

- Output is assigned to average:** Points to the variable `average` on the left side of the assignment.
- Function Name:** Points to the function name `average2` in the call.
- Input:** Points to the arguments `100` and `50` in the call.

Feb 2, 2018

Sprenkle - CSCI111

`average2.py`

4

Review: 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

Feb 2, 2018

Sprenkle - CSCI111

5

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

Feb 2, 2018

Sprenkle - CSCI111

6

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)
 - Otherwise, unintended side effect of calling the function
 - When you have a function that is supposed to display something
 - Sometimes, that is what you want.

Feb 2, 2018

Sprenkle - CSCI111

7

Review: 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

Feb 2, 2018

Sprenkle - CSCI111

8

Review: Example program with a main()

```
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()
```

Constants, comments
are in example program

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

oldmac.py

Review: Example program with a main()

```
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()
```

1. Set definition of main
2. Set definition of printVerse
3. Call main function
4. Execute main function
5. Call, execute printVerse

...

oldmac.py

VARIABLE LIFETIMES AND SCOPE

Feb 2, 2018

Sprenkle - CSCI111

11

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()
```

Feb 2, 2018

Sprenkle - CSCI111

mystery.py

12

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**?

Feb 2, 2018

Sprenkle - CSCI111

mystery.py

13

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

Feb 2, 2018

Sprenkle - CSCI111

14

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()

Memory stack

main	x	10
------	---	----

Variable names
are like first names

Function names are like last names

Define the **SCOPE** of the variable

Feb 2, 2018

Sprenkle - CSCI111

15

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

sum Evens	limit	10
main	x	10

Feb 2, 2018

Sprenkle - CSCI111

16

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()
```

sum Evens	total 0 limit 10
main	x 10

Feb 2, 2018

Sprenkle - CSCI111

17

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()
```

sum Evens	x 0 total 0 limit 10
main	x 10

Feb 2, 2018

Sprenkle - CSCI111

18

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()
```

sum Evens	x 8 total 20 limit 10
main	x 10

Feb 2, 2018

Sprenkle - CSCI111

19

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

main	sum 20 x 10
------	----------------

Feb 2, 2018

Sprenkle - CSCI111

20

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()
```

main	x 10 sum 20
------	----------------

Feb 2, 2018

Sprenkle - CSCI111

21

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

Feb 2, 2018

Sprenkle - CSCI111

22

Summary: Why Write Functions?

- Allows you to break up a hard 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*)

Similar to benefits of OO Programming

Feb 2, 2018

Sprenkle - CSCI111

23

TESTING FUNCTIONS

Feb 2, 2018

Sprenkle - CSCI111

24

Testing Functions

- Functions make it easier for us to test our code
- We can write code to test the functions
 - Test Case:
 - Input: parameters
 - Expected Output: what we expect to be returned
 - We can verify the function programmatically
 - “programmatically” – automatically execute test cases and verify that the actual returned result is what we expected
 - No user input required!

Feb 2, 2018

Sprenkle - CSCI111

25

test Module

- FUNCTIONS
 - `testEqual(actual, expected)`

Feb 2, 2018

Sprenkle - CSCI111

26

Example: Testing sumEvens

```
import test
...
def testSumEvens():
    actual = sumEvens( 10 )
    expected = 20
    test.assertEqual( actual, expected )

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

This is the actual result from our function

This is what we expect the result to be

What are other good test cases?

testSumEvens.py

Feb 2, 2018

Sprenkle - CSCI111

27

Broader Issue: Net Neutrality

Anna Ben Kalady Lindsey Olivia	Harris Ian Parker Rachel Robert	Alison Chase Lizzie Mary-Frances	Andrew Davis Findley Ryan	Chas Jordan Joseph Margaret
--	---	---	------------------------------------	--------------------------------------

Feb 2, 2018

Sprenkle - CSCI111

28

Net Neutrality

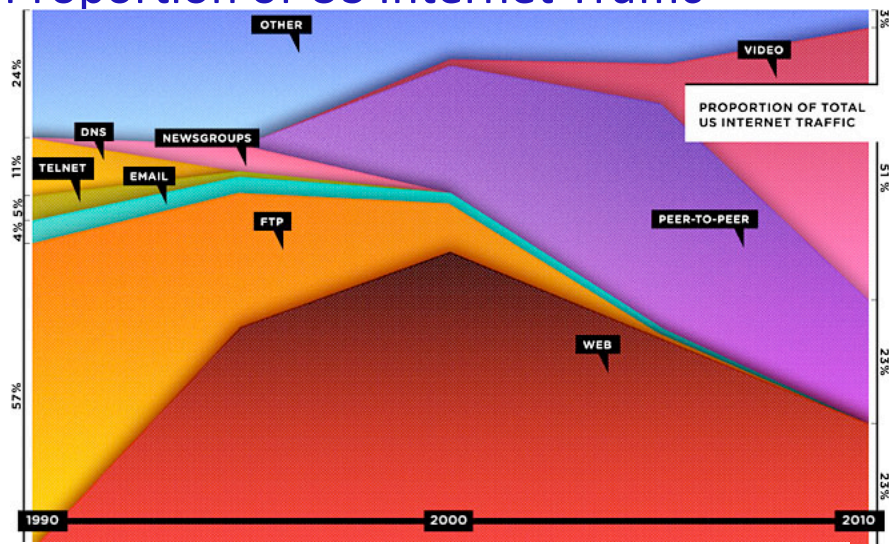
- What is net neutrality?
- Is this an issue?
 - Argument: hasn't been an issue up until now
- What are the arguments for/against net neutrality?
 - Who are the stakeholders in net neutrality?
 - What are their takes?
 - "My view is that the Internet should be run by engineers and entrepreneurs, not lawyers and bureaucrats." – Ajit Pai, FCC head
- How is this similar/different to phone calls or TV?

Feb 2, 2018

Sprenkle - CSCI111

29

Proportion of US Internet Traffic



Sources: Cisco estimates based on CAIDA publications

Andrew Odlyzko

https://www.wired.com/2010/08/ff_webrip/