CSCI 141

Lecture 15
Functions:
More Scope, Return Values
Tuples
Announcements
Announcements

• A4 out today, due Friday 5/17
Announcements

• A4 out today, due Friday 5/17

• Midterm grades should be out Thursday night
Announcements

• A4 out today, due Friday 5/17

• Midterm grades should be out Thursday night

  • Released via Gradescope: you’ll receive an email with instructions on how to login and see your feedback.
Announcements

- A4 out today, due Friday 5/17
- Midterm grades should be out Thursday night
  - Released via Gradescope: you’ll receive an email with instructions on how to login and see your feedback.
- Last lecture’s worksheet Exercise 3 has a typo:
Announcements

• A4 out today, due Friday 5/17

• Midterm grades should be out Thursday night
  • Released via Gradescope: you’ll receive an email with instructions on how to login and see your feedback.

• Last lecture’s worksheet Exercise 3 has a typo:
  • Should say: Defines a function that takes a single argument and prints the fourth power of the input argument.
Goals

• Know how to use **parameters** to refer to the input arguments of a function

• Know the meaning of **local variables** and **variable scope** and how it relates to function parameters.

• Know how to **return** a value from a function, and the behavior of the return statement.

• Understand the basic usage of **tuples**:  
  • using tuples to return multiple values from a function  
  • **packing** and **unpacking** via assignment
Parameters vs Arguments

Parameters: variable names that will refer to the input arguments.

Parameters (these are new):
variables that take on the value of the arguments

def add2(a, b):
    """ Print the sum of a and b """
    print(a + b)

add2(4, 10)

Arguments (we’ve seen these before):
values passed into a function.
Parameters are Local Variables

- They **only** exist inside the function.
- Any other variables declared inside a function are also local variables.
- This is an example of a broader concept called **scope**: a variable’s scope is the set of statements in which it is visible/usable.
- A local variable’s scope is limited to the function inside which it’s defined.
Function Calls: A Model for Execution

```python
def axpy(a, x, y):
    """ Print a * x + y """
    product = a * x
    result = product + y
    print(result)

a1 = 2
x1 = 3
print(axpy(a1, x1, 4))
```

1. Evaluate all arguments
2. Assign argument values to parameter variables
3. Execute the function body
4. When done, replace the function call with its return value.
Demo via add2
Demo via add2

• Using Thonny’s debug mode to see the local variables inside the scope of a function:
  • passing in values
  • passing in variables, which evaluate to values that get assigned to the parameters
  • passing in global variables with the same name, which get **shadowed** by the local variables
def print_rectangle_area(width, height):
    """ Print the area of a width-by-height rectangle """

    area = width * height
    print(area)

w = 4
h = 3
a = w * h
print_rectangle_area(w, h)
def print_rectangle_area(width, height):
    """ Print the area of a width-by-height rectangle """

    area = width * height
    print(area)

w = 4
h = 3
a = w * h
print_rectangle_area(w, h)
```python
def print_rectangle_area(width, height):
    """ Print the area of a width-by-height rectangle """

    area = width * height
    print(area)

w = 4
h = 3
a = w * h
print_rectangle_area(w, h)
```
Variable Scope

```python
def print_rectangle_area(width, height):
    """ Print the area of a width-by-height rectangle """

    area = width * height
    print(area)

w = 4
h = 3
a = w * h
print_rectangle_area(w, h)
```

Which version of line 12 does not do the same thing as line 11?

A. print(h * w)
B. print(width * height)
C. print(w * h)
D. print_rectangle_area(h, w)
def print_rectangle_area(width, height):
    """ Print the area of a width-by-height rectangle """

    area = width * height
    print(area)

w = 4
h = 3
a = w * h
print_rectangle_area(w, h)

What if I want to do further computation with the result of the rectangle area?
def print_rectangle_area(width, height):
    """ Print the area of a width-by-height rectangle """

    area = width * height
    print(area)

w = 4
h = 3
a = w * h
print_rectangle_area(w, h)

What if I want to do further computation with the result of the rectangle area?
It got printed, then it was gone…
Writing Functions: Syntax

```python
def name(parameters):
    statements
```

Two important questions:
1. How does the function use the arguments (inputs) passed to it?
2. How does the function return a value?
Returning values

New statement: the `return` statement

Syntax: `return` expression

Behavior:

1. `expression` is evaluated
2. the function stops executing further statements
3. the value of expression is returned
   i.e., the function call `evaluates` to the returned value
Returning values

New statement: the `return` statement

Syntax: `return expression` (can only appear inside a function definition)

Behavior:

1. `expression` is evaluated

2. the function stops executing further statements

3. the value of `expression` is returned
   i.e., the function call `evaluates` to the returned value
Demo: Make add2 return instead of print
**Function Syntax: Summary**

```
def name(parameters):
    """ docstring """
    statements
    inputs
    comma-separated list of parameters: variable names that will get assigned to the arguments
    An indented code block that does any computation, executes any effects, and (optionally) returns a value
    effects; return value
```
Today's Quiz

• 3 minutes

• Math reminder:

\[ (p_1^x, p_1^y) \quad \text{and} \quad (p_2^x, p_2^y) \]

\[ a = p_2^x - p_1^x \]
\[ b = p_2^y - p_1^y \]
\[ c = \sqrt{a^2 + b^2} \]
Today’s Quiz

• 3 minutes

• Working with a neighbor: do your answers agree? (2 minutes)
Distance Function: Demo
Why write functions?

• The convenience of repetition:
  • you can define a function once then call it as many times as you want

• The power of customized repetition:
  • you can define a function that takes arguments to customize the task it performs: this is powerful!
  • e.g.: one function to draw any size rectangle, or any n-sided polygon

• The power of function composition.
  • Functions can call other functions.
Returning values

New statement: the `return` statement

Syntax: `return expression`

Behavior:

1. `expression` is evaluated

2. the function stops executing further statements

3. the value of `expression` is returned
   i.e., the function call `evaluates` to the returned value
Returning values

New statement: the `return` statement

Syntax: `return expression` (can only appear inside a function definition)

Behavior:

1. `expression` is evaluated

2. the function stops executing further statements

3. the value of expression is returned
   i.e., the function call `evaluates` to the returned value
Returning Early: Demo

def sign(x):
    """ Return -1 if x < 0,
          1 if x > 0,
       or 0 if x == 0 """

    # code here
def midpoint(p1x, p1y, p2x, p2y):
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """
    # code here
    (mid_x, mid_y)

This is two things!? Can we return two things?
def midpoint(p1x, p1y, p2x, p2y):
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """

    # code here

(mid_x, mid_y)

This is **two** things!? Can we return two things?
Midpoint Function

def midpoint(p1x, p1y, p2x, p2y):
    
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """

    # code here
    # mid_x = . . .
    # mid_y = . . .
def midpoint(p1x, p1y, p2x, p2y):
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """

    # code here
    # mid_x = . . .
    # mid_y = . . .

    return mid_x, mid_y
Midpoint Function

# mid_x = . . .
# mid_y = . . .

Okay, but how do you actually calculate this?

(on the board)
Midpoint Function

# mid_x = . . .
# mid_y = . . .

Okay, but how do you actually calculate this?

\[(p2_x, p2_y)\]

\[(mid_x, mid_y)\]

\[(p1_x, p1_y)\]

(mid_x, mid_y)

mid_x

mid_y

(on the board)

\[mid_x = (p1_x + p2_x) / 2\]

\[mid_y = (p1_y + p2_y) / 2\]
Returning Multiple Values

- You can return multiple values from a function by grouping them into a comma-separated sequence:
  
  ```
  return mid_x, mid_y
  ```

- You can assign each to a variable when calling the function:
  
  ```
  mx, my = midpoint(plx, ply, p2x, p2y)
  ```
These are actually **tuples**

- A tuple is a sequence of values, optionally enclosed in parens.
  
  \[(1, 4, \text{"Mufasa"})\]

- You can “pack” and “unpack” them using assignment statements:
  
  ```python
  v = (1, 4, \text{"Mufasa"})
  
  (a, b, c) = v
  ```
These are actually tuples

- Tuples can also be passed into functions as arguments:

```python
def midpoint(p1, p2):
    """Compute the midpoint between p1 and p2"""
    p1x, p1y = p1
    p2x, p2y = p2

    # . . .
    # return mx, my
```
Tuples: Demo