Functions

Chapter 3

"Conjunction junction what's your… "

Schoolhouse Rock, anybody?

Today's Outline

- Functions
  - Built in functions
  - Defining new functions
- Execution Flow (multiple functions)
- Function Arguments
- Returned values
- Write code for some new functions!

Functions

- What is a "function?"
  - A *named* sequence of statements that performs a computation.
- We've already seen some "Built in" functions:
  - `type(3.14)`
    `<type 'float'>`
  - `print("Hello World!")`
- "Type" is the name of the function. The expression is in parenthesis (3.14)and is the "argument."

Type Conversion Functions

- Use `int`, `float`, and `str` to convert values from one type to another.
  - This is also why you can't assign a variable called "int"
- When might you do this?
  - "And number 15 comes on the field to a raucous round of approval from the stands."
  - That's US soccer great, Megan Rapinoe.
  - In this case it's a *string* and not an *integer*.

Type Conversion Functions

- Use `int`, `float`, and `str` to convert values from one type to another.
- Example
  - `What's the difference between '15' and 15 without quotes?`

How can we check to make sure?

- `Rapinoe = 15 #Pronounced Rah-PEE-no.`
- `type(Rapinoe)`
- `<type 'int'>`

Type Conversion

- Use int, float, and str to convert values from one type to another.
- Example continued
  - `float(32) # converts an integer into a float: what is the result?`
    `32.0`
  - `str(42) # converts an integer into a string "42"`
  - `str(3.14149) # converts a float into a string`

Type Conversion

- Can be used within expressions
- Example
  - 59 / 60
    0 **#Result in Python 2! What about Python 3?**
  - number = 59
    number / 60.0 #What's the result?
    0.9833333333333333
  - number = 59
    float(number) / 60 # converts to float
    0.9833333333333333

Modules

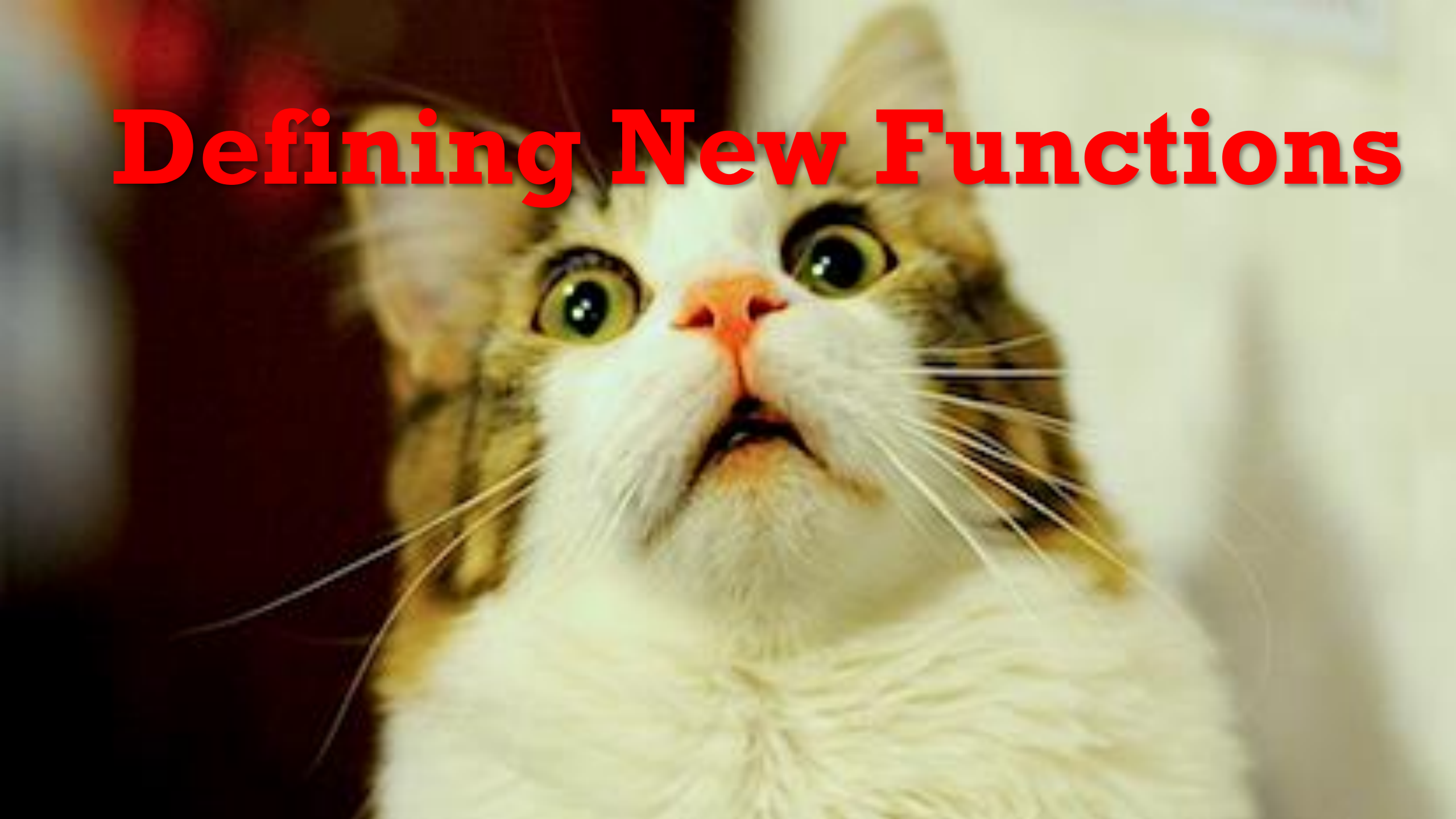
- Many math functions can be found in module called "math"
  - A file that contains a collection of related functions. You need to import these.
- Look at documentation for help!
  - https://docs.python.org/2/py-modindex.html
  - https://docs.python.org/3/py-modindex.html
- Over time modules may change or be "deprecated" meaning they are not suggested for use.

Modules

- Import the math module
  - `import math # math is the name of a module.`
- To access a function in a module you need to specify the name of the module, followed by a period.
  - `math.pi # What does this function do?`
- Use the `help` function to see what things do.
  - `help(math.log) #What's the result?`

DOCUMENTATION!!!!

Defining New Functions



Defining New Functions

- This will significantly boost your coding skillz!
- Creating clean, repeatable code, is important.
- Functions allow you create blocks of code that can be repeated many times versus just copying and pasting them line by line.
- Functions can also give you outputs that can feed into more functions.

Defining New Functions

- Syntax

```
def NAME (ARGUMENTS): # Empty () means no arguments
    STATEMENTS
```

- Example

```
# This function prints a string
>>> def print_strings():
        print "Python Programming"
        print "Super cool"

# Call function
print_strings ()
```

Explaining New Functions

- It's always a good idea to explain what your function is doing.
- Syntax:

```
def print_string ():
        # DOCSTRING: information about the function.
        # Expected input is WHATEVER.
        # Expected output is WHATEVER.
   print "Python version 3.0"
# Call function
print_string ()
```

Functions: Execution Flow

- What's the output and **order** of this code?

```
def WorkdayHours():
      print("Mon-Fri: 9-9")
def WeekendHours ():
      print("Weekends: 11 - 5")

#Call two previously defined functions
def ShopHours():
      WorkdayHours()
      WeekendHours()

# A function call
ShopHours()
```

1

2

3

Function Arguments

- Some functions require "arguments." E.g.
  - `Math.sin(2)` or `print_twice(bruce)`
- A parameter is a variable in a method definition. When a method is called, the *arguments* are the *data* you pass into the method's parameters.
- The argument is the actual value of the variable that gets passed to the function.
  - `def print_twice(bruce) # "bruce" is the parameter.`

Function Arguments

- A parameter is a variable
- *arguments* are the *data* you pass into the method's parameters.
- These two things can be different. Think of the argument as a placeholder for the actual input argument.

Function Arguments

- Variables created in functions are *local*: they only exist in that function. Parameters are also local.
- Outside of `print_twice` there is no `bruce`, which is the parameter.

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
# Try running "bruce"
bruce
# Now try running the function w/ arguments.
print_twice(42)
```

Function Arguments

- This codes prints something twice. Create a function, that calls this function, to print something 4 times.

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
print_twice(42)
```

Answer

```
def print_four(bruce):
    print_twice(bruce)
    print_twice(bruce)
print_four(42)
```

Fruitful vs. Void

- What's the difference between a "fruitful function" and a "void function?"
  - Void functions perform an action, but do not return return a value. Our previous code just printed what we told it to.
  - Fruitful functions return a value (e.g. a math problem).
- A *return* statement ends the execution of the function call and "returns" the result ("the fruit"), i.e. the value of the expression following the return keyword, to the caller.

Returned Values

- What's the output and order of this code?

```
def square(x):
    y = x * x
    return y
    #Press return again to get a space!
square(7)
```

Returned Values

- What's the output and order of this code?

```
def square(x):
    y = x * x
    return y
    #Press return again to get a space!

toSquare = 10
result = square(toSquare)
print("The result of ", toSquare, "squared is", result)
```

Exercise 1

Create and run a .py program to calculate the area of a circle.

1. What is the formula?
   1. The area of a circle is pi times the radius squared ($A = \pi r^2$).
2. The function takes radius as its input parameter, and returns the area.
3. Print the calculated area by calling the function.

Exercise 1

Example answer:

```
import math
def circle_area (r):
      #Function defines area of a circle.
      #Input: radius
      a = r**2 * math.pi
      return a

 print(circle_area(r))
```

Exercise 2

Make your .py program interactive by using the "input" function.

1. Syntax: `input()`
2. Using the previous code, create a line that makes r equal to input from the user.

```
r = input()
```

3. Use type conversion to make sure integers are converted to floats, in case the users enters a whole number.

Exercise 2

Example answer:

```
import math
def circle_area (r):
        """
        Function defines area of a circle.
        Input: script prompts users to enter value for r
        """
        r = float(input ("Input the radius of the circle : "))
        a = r**2 * math.pi
        return a

 print(circle_area(r))
```

Summary

- Built in functions exist (e.g. print)
- Follow the syntax for defining new functions
- Execution Flow is important (and can be tricky!)
- Function Arguments
- Returned values
- Practice Defining some new functions

Readings for Next Week

- Chapter 4 sections
  - “Simple Repetition”
  - “A Development Plan”
  - “Debugging”
- Ch. 5 Conditionals & Recursions