FILES AND FUNCTIONS

CSSE 120 – Rose-Hulman Institute of Technology

Eclipse Project

- Make an Eclipse PyDev Project called:
  - Session07_FilesAndFunctions

- Then right click on the src folder to make new PyDev modules for each practice exercises during today's class
- **Reminder:** When naming a new PyDev module, in Eclipse, you do not include the **.py** extension; Eclipse adds that automatically for you.

File Processing

- Manipulating data stored on disk
- Key steps:
  - *Open* file
    - For reading or writing
    - Associates file on disk with a *file variable* in program
  - *Manipulate* file with operations on file variable
    - Read or write information
  - *Close* file
    - Causes final "bookkeeping" to happen

File Writing in Python

- Open file:
  - Syntax: <filevar> = open(<name>, <mode>)
  - Example: outFile = open('average.txt', 'w')
    - Replaces contents!
- Write to file:
  - Syntax: <filevar>.write(<string>)
- Close file:
  - Syntax: <filevar>.close()
  - Example: outFile.close()

File Reading in Python

- Open file: inFile = open('grades.txt', '**r**')
- Read file:
  - <filevar>.read() — Returns one **BIG** string
  - <filevar>.readline() — Returns next line, including \n
  - <filevar>.readlines() — Returns **BIG** list of strings, 1 per line
  - for <ind> in <filevar> — Iterates over lines efficiently
- Close file: inFile.close()

- Create a program that reads and prints itself

A “Big” Difference

- Consider:
  - inFile = open ('grades.txt', 'r‘)
    for line in inFile.readlines():
        # process line
    inFile.close()
  - inFile = open ('grades.txt', 'r‘)
    for line in inFile:
        # process line
    inFile.close()
- Which takes the least memory?

Q2

Your turn

- Make a new PyDev Module called numberWrite.py
- Write code to print the numbers 1-100 to a file named ‘numbers.txt,’one per line.

Why functions?

- A function allows us to group together several statements and give them a name by which they may be invoked.
  - Abstraction (easier to remember the name than the code)
  - Compactness (avoids duplicate code)
  - Flexibility (parameters allow variation)
- Example:

```
def complain(complaint):
    print "Customer:", complaint
```

Functions in different realms

We compare the mechanisms for defining and invoking functions in three different settings:

- Standard mathematical notation
- Maple
- Python

Functions in Mathematics

- Define a function:
  - $f(x) = x^2 - 5$

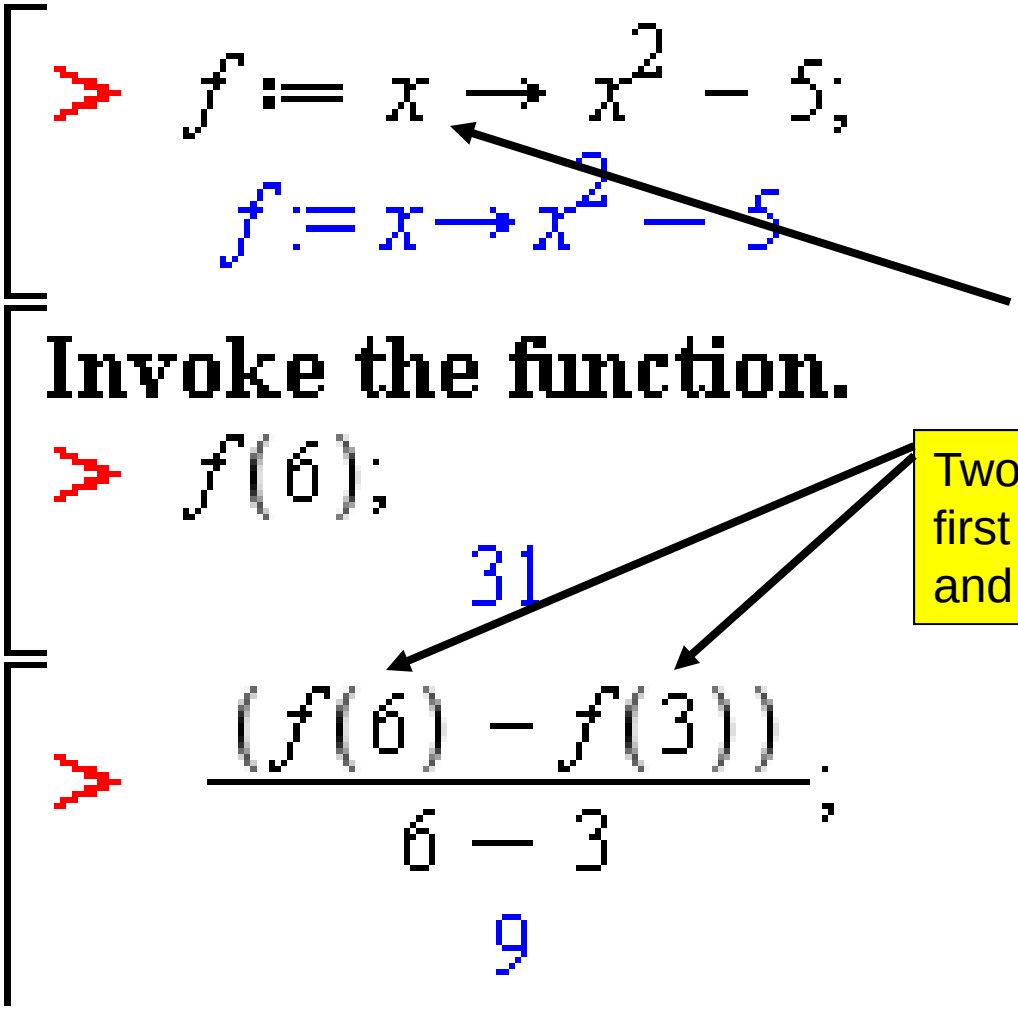
Formal Parameter. Used so that we have a name to use for the argument in the function's formula.

- Invoke (call) the function:
  - $\dfrac{f(6) - f(3)}{6 - 3}$

Two calls to function **f**. The first with actual parameter 6, and the second with 3.

- When the call f(6) is made, the actual parameter 6 is substituted for the formal parameter x, so that the value is $6^2 - 5$.

Functions in Maple



Formal Parameter. Used so that we have a name to use for the argument in the function's formula.

Two calls to function **f**. The first with actual parameter 6, and the second with 3.

Functions in Python

```
>>> def f(x):
        return x*x - 5

>>> f(6)
31
>>> (f(6) - f(3)) / (6 - 3)
9
>>>
```

Formal Parameter. Used so that we have a name to use for the argument in the function's formula.

Two calls to function **f**. The first with actual parameter 6, and the second with 3.

- How would you evaluate **f(f(2))**?
- In Mathematics, functions calculate a value.
- In Python we can also define functions that instead *do something*, such as print some values.

Review: Parts of a Function Definition

```
>>> def hello():
        print "Hello"
        print "I'd like to complain about this parrot"
```

Defining a function called “hello”

Indenting tells interpreter that these lines are part of the hello function

Blank line tells interpreter that we’re done defining the hello function

Review: Defining vs. Invoking

- Defining a function says what the function should do
- Invoking a function makes that happen
  - Parentheses tell interpreter to invoke the function

```
>>> hello()
Hello
I'd like to complain about this parrot
```

Review: Function with a Parameter

- def complain(complaint):

```
    print "Customer: I purchased this parrot not half " +
          "an hour ago from this very boutique"
    print "Owner: Oh yes, the Norwegian Blue. " +
         " What's wrong with it?"
    print "Customer:", complaint
```

- invocation:
  - complain("It's dead!")

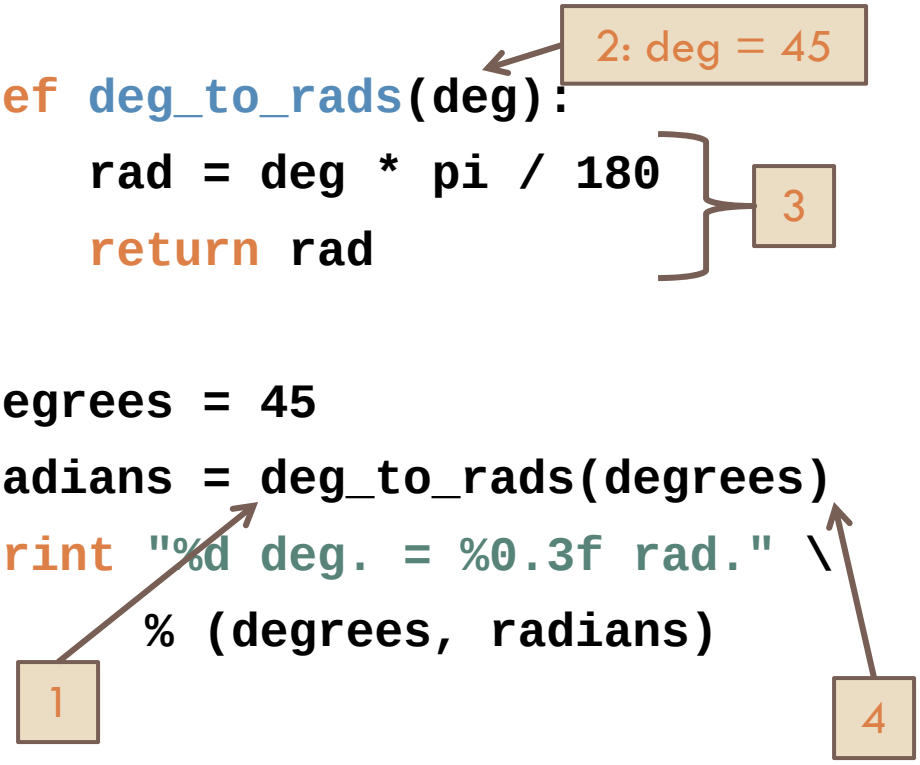
When a function is invoked (called), Python follows a four-step process:

1. Calling program pauses at the point of the call
2. Formal parameters get assigned the values supplied by the actual parameters
3. Body of the function is executed
4. Control returns to the point in calling program just after where the function was called

```
from math import pi


def deg_to_rads(deg):
    rad = deg * pi / 180
    return rad


degrees = 45
radians = deg_to_rads(degrees)
print "%d deg. = %0.3f rad." \
      % (degrees, radians)
```



Functions can (and often should) return values

- We've **written** functions that just do things
  - hello()
  - complain(complaint)
- We've **used** functions that *return* values
  - abs(-1)
  - fn_root_1 = math.sqrt(b*b – 4*a*c)
- Define a function that returns a value

```
def square(x):
    return x * x
```

return statement

Why might it be better to **return** than **print** when a function performs a calculation?

Exercise – writing a `distance()` function

- Make a new PyDev Module called distance.py
- Go to Angel to grab some starting code
  - Lessons -> Modules to download in class -> Session 7
- Copy the contents into your distance.py
- Write and test a `distance` function:

```
def distance(p1, p2):
    """Parameters are Points, returns distance between them."""
```

- Should the function return anything?

If a Function Calls a Function …

```
def g(a,b):
    print a+b, a-b

def f(x, y):
    g(x, y)
    g(x+1, y-1)

f(10, 6)
```

- Trace what happens when the last line of this code executes
- Now do the **similar** one on the quiz

Q9

An exercise in code reading

- With a partner, read and try to understand the code that is on the handout.
- You can probably guess what the output will be. But how does it work?
- Figure that out, discuss it with your partner and answer quiz question 10.
- Optional Challenge Problem for later: try to write "There's a Hole in the Bottom of the Sea" or "The Green Grass Grew All Around" in a similar style.
- When you are done, turn in your quiz and start HW