As you arrive:

1. Start up your computer and plug it in
2. ***Log into Angel*** and go to CSSE 120
3. Do the ***Attendance Widget*** – the PIN is on the board
4. Go to the course ***Schedule Page***
5. Open the ***Slides*** for today if you wish
6. Check out today's project: `09-MoreFunctions`

Plus in-class time working on these concepts AND practicing previous concepts, continued as homework.

Functions, revisited

- Defining, parameters
- Calling, actual arguments
- Returning values

Functions, new

- Optional parameters
- Returning multiple values (tuples)
- Mutators

Checkout today's project: `09-MoreFunctions`

Troubles getting today's project? If so: →

Are you in the Pydev perspective? If not:

- `Window ~ Open Perspective ~ Other` then `Pydev`

Messed up views? If so:

- `Window ~ Reset Perspective`

No SVN repositories view (tab)? If it is not there:

- `Window ~ Show View ~ Other` then `SVN ~ SVN Repositories`

In your SVN repositories view (tab), expand your repository *(the top-level item) if not already expanded.*

- If no repository, perhaps you are in the wrong Workspace. Get help as needed.

Right-click on today's project, *then select* **`Checkout`***. Press* **`OK`** *as needed.*

The project shows up in the **`Pydev Package Explorer`** to the right. Expand and browse the modules under **`src`** as desired.

Outline of Today's Session

- Questions?
- Functions, review
- Functions, ***new ideas***
  - ***Optional parameters***
  - ***Returning multiple values*** from a function
    - By returning a ***tuple***
  - ***Mutators***
    - Functions that modify the characteristics of their parameters

Checkout today's project:
`09-MoreFunctions`

Practice, practice, practice!

- Functions
- Lists and List methods
- Strings and String methods
- Using objects
  - Zellegraphics
- Definite loops
- ***Pair programming***

Functions – Outline

Check out today's project:
`09-MoreFunctions`

- ***Functions, review***
  - Why use functions?
    - Abstraction
    - Compactness
    - Flexibility / Power
  - ***Defining*** a function
    - *Parameters*
  - ***Calling*** (***invoking***) a function
    - *Actual arguments*
    - What happens – 4 steps

- ***Functions, new***
  - *Optional* parameters
  - *Returning multiple values*
    - by returning a *tuple*
  - *Mutators*

- ***Returning*** values from a function
  - *Return* statement
  - Capturing the returned value in a variable

Review: Why functions?

- A function allows us to:
  - group together several statements,
  - give them a name by which they may be invoked, and
  - supply various actual arguments that get used in the function body as the values of the formal parameters.
- As such, functions have three virtues:
  - ***Abstraction***
    - It is easier to remember and use the *name* than the code.
  - ***Compactness***
    - Functions help you avoid duplicate code.
  - ***Flexibility / Power***
    - The parameters allow variation – the function can do *many* things, depending on the actual arguments supplied.

Example on the next slide

Review: Why functions? 3 virtues:

- ***Abstraction***
  - It is easier to remember and use the *name* than the code.
- ***Compactness***
  - They let you avoid duplicate code.
- ***Flexibility/ Power***
  - The parameters allow variation – the function can do *many* things, depending on the actual arguments supplied.

Example:

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

def nastyPeopleSayThingsLike():
    complain("Bob", "Your store stinks.")
    complain("Alice", "You stink.")
    complain("Letterman", "Your jokes stink.")
    complain("Letterman", "Your jokes stink.")
    complain("Letterman", "Your jokes stink.")
```

Abstraction

Flexibility / Power

Compactness

Q1

Review: *Defining* vs. *Calling* (*Invoking*)

- ***Defining*** a function ***says*** what the function should do

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

- ***Calling*** (***invoking***) a function ***makes*** that happen
  - Parentheses tell interpreter to *call* (aka *invoke*) the function

```
hello()

Hello.
I'd like to complain about this parrot.
```

Q2

Review: *Parameters* vs. *Actual arguments*

```
def squareNext(x):
    '''
        Returns the square of the number one bigger than the given number,
        that is, returns the square of the "next" number.
    '''
    x = x + 1        # Bad form
    return  x * x

def main():
    y = 3
    answer = squareNext(y)
    print(y, answer)

    x = 8
    answer = squareNext(x)
    print(x, answer)

    x = 10
    answer = squareNext(x + 19)
    print(x, answer)
```

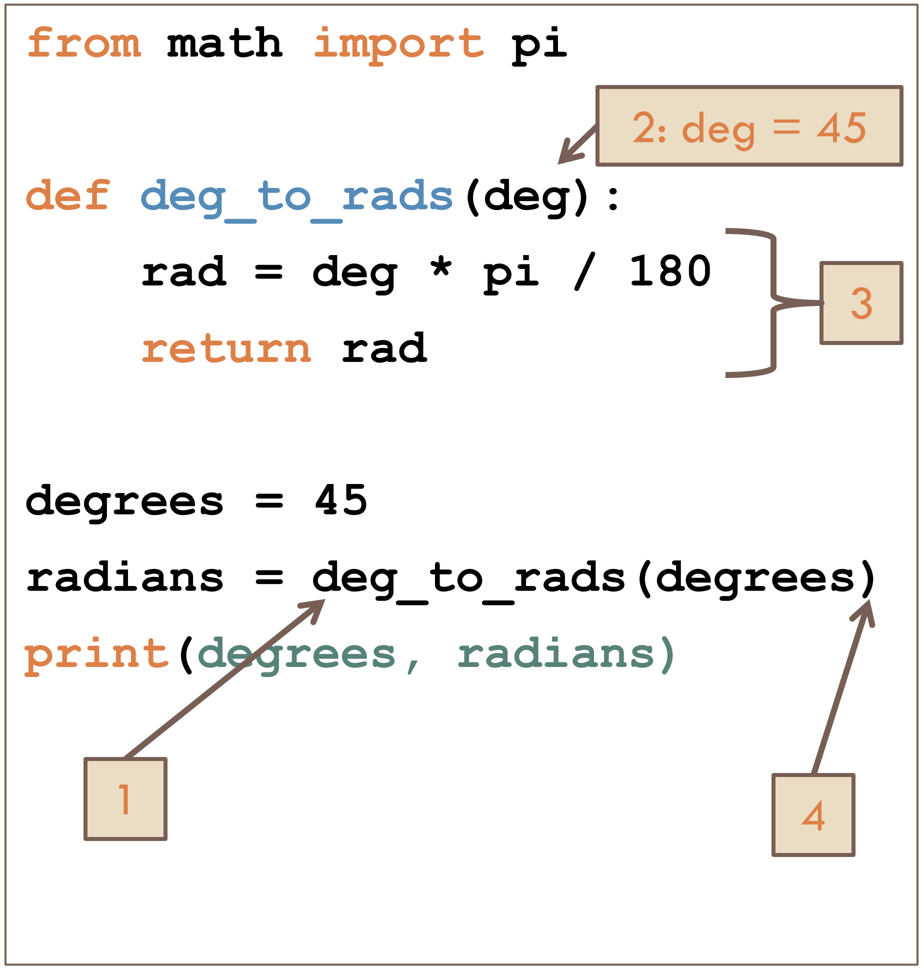
Do the exercise in the
`1-actualArguments.py`
module in today's project:

1. ***Examine*** the `squareNext` and `main` functions.
2. ***Predict*** what will be printed when `main` runs.
3. ***Run*** the module. Was you prediction correct?
4. Answer the quiz question.
   Ask questions as needed.

Q3

Review: The ***4-step process*** when a function is ***called*** (aka ***invoked***)

1. Calling program pauses at the point of the call.
2. Formal parameters get assigned the values supplied by the actual arguments.
3. Body of the function is executed.
   - The function may *return* a value.
4. Control returns to the point in calling program just after where the function was called.
   - If the function returned a value, we capture it in a variable or use it directly.



```
from math import pi

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


degrees = 45
radians = deg_to_rads(degrees)
print(degrees, radians)
```

Review – ***Returning*** a value from a function

```python
def factorial(n):
    ''' Returns n!.  That is, returns n * (n-1) * (n-2) * ... * 1.
        Returns 0 if n < 1.  Assumes n is an integer.
    '''
    product = 1
    for k in range(1, n + 1):
        product = product * k

    return product

def main():
    ''' Prints a table of factorial values. '''
    for k in range(21):
        kFactorial = factorial(k)
        print("{}! is {}".format(k, kFactorial))
```

return statement

Leaves the function and sends back the returned value.

Capture the returned value in a variable.

Or, use it directly (e.g., in the *print* statement).

Q4

Functions, ***new ideas*** – Outline

- ***Functions, review***
  - Why use functions?
    - Abstraction
    - Compactness
    - Flexibility / Power
  - ***Defining*** a function
    - *Parameters*
  - ***Calling*** (***invoking***) a function
    - *Actual arguments*
    - What happens – 4 steps

- ***Functions, new***
  - *Optional* parameters
  - *Returning multiple values*
    - by returning a *tuple*
  - *Mutators*

- ***Returning*** values from a function
  - *Return* statement
  - Capturing the returned value in a variable

Optional parameters

- A python function may have some parameters that are optional.

```
>>> int("37")
37
>>> int("37", 10)
37
>>> int("37", 8)  # specify base 8
31
```

Also look at calls to GraphWin

We can declare a parameter to be optional by supplying a default value.

```
>>> def printDate(month, day, year=2007):
        print month, str(day)+",", year


>>> printDate("January", 4, 2006)
January 4, 2006
>>> printDate("January", 4)
January 4, 2007
```

Multiple optional parameters

- If there are more than one, and it's not clear which argument you are providing, you can pass **variable=value**:

Note that all 3 are optional:

```
>>> def printDate(month = 'January', day = 1, year=2007):
                print month, str(day)+',', year
```

```
>>> printDate()
January 1, 2007
>>> printDate(26)
26 1, 2007
>>> printDate(day=26)
January 26, 2007
```

Nice!

I wanted the 26th. Whoops!

That's it.

Q5

Returning Multiple Values

- A function can return *multiple* values

```
def powers(n):
    return n**2, n**3, n**4
```

- What is the *type* of the value returned by this call?

```
powers(4)
```

  - Answer: it is a *tuple*

- In the caller, how do you *capture* the returned tuple?
  - Assign returned values individually, or to a *tuple*:

```
p2, p3, p4 = powers(5)
listOfPowers = powers(5)
```

Q6

Mutators: Passing a *mutable* parameter

- Functions can change the *contents* of a *mutable* parameter. Such functions are called ***mutators***.

```
def addOneToAll(listOfNums):
    for i in range(len(listOfNums)):
        listOfNums[ i ] +=1

def main():
    myList = [1, 3, 5, 7]
    addOneToAll(myList)
    print(myList)
```

- What does this print?
  What actually gets passed to the function?

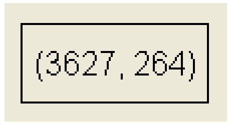
Q7-9, *turn in quiz*

Homework

- Some parts are not easy; we suggest that you start it today so you can get help during assistant lab hours this afternoon or evening if you get stuck.
- After you finish `threeSquares`, work on `triangles` until the end of class.
- If you also finish `triangles`, work on the other parts of the homework.

Pair Programming: Three Squares

1. Run the threeSquares program to be sure it works. Put **both** students' names in the initial comment.
2. Add a function, **stats**, that takes a Rectangle, **r**, as a parameter and returns the area of **r**
3. modify the program so that it displays the area of each rectangle inside the rectangle
4. Finally, change **stats** to return the area and perimeter (see figure at right)
5. Commit your project back to your repository; also email **threeSquares.py** to your partner.



```
(3627, 264)
```

Example Display

Rest of session

- Continue your homework:
  - Homework 8 due Wednesday.
  - Homework 9 due Thursday.