Announcements

- The grades of Lab01 and Lab02 are out.
- Project 1 will be out on Friday 26\textsuperscript{th} and due on Oct. 12\textsuperscript{th}
- Midterm 1 is on Oct 9\textsuperscript{th}, 6:30-7:30 PM in CL50 224
  - Let us know in advance about conflicts or other valid makeup excuses.
Functions

- Previously, we have used function: range, eval, sqrt, etc.
- Functions allow grouping statements together under a (function name) that can be executed by calling the function
- Functions are a collection of instructions that perform a task as:
  - Printing your name and course
  - Calculating the average of a set of numbers
  - Editing a picture or video
Functions

• Like in algebra, a function is a kind of “box” into which you put one value and out comes another. We represent ("denote") a function by a name (in math we use f or F).
Why to use a function?

- If we define a function to perform a task, then we will write it once but we can use it (or call it) many times.
- Functions can make a program easier to read and debug.
- Functions can make a program shorter as their use can eliminate repetitive code.
- Functions allow that future changes need to be only made in one place.
- Dividing a program into functions allows one to debug parts one at a time.
- Well-designed functions are often useful in other programs and can allow the reuse of code.
How to write functions?

```python
def functionName():
    statement #1
    statement #2
...
```

- Indentation is very important in Python, it marks the beginning of function body
- Python will give errors if your function is not properly indented

*It is a programming practice to define a function that is called main to call the other functions in our program*
def SayHello():
    print("Hello world!")
    print("--From Python")

Note: 1. Don’t forget the colon(:)

2. Align the statements in one function
Functions: Arguments

- A function may or may not receive one or more argument

**No Argument**

```python
def Greet():
    print("Hello Jack")

def main():
    Greet()
```

**One Argument**

```python
def GreetWithArg(message):
    print(message)

def main():
    msg = "Hello Jack"
    GreetWithArg(msg)
```
Functions: Arguments

- A function argument can be:
  1. A value
  2. Expression
  3. A variable

```python
def Sum(a, b):
    total = a + b
    print(total)

def main():
    x = 5
    y = 10
    Sum(4, 10)
    Sum(x+2, y-3)
    Sum(x, y)
```

Arguments are values
Arguments are expressions
Arguments are variables
Function: Arguments

- On function call, Python assigns the value of the argument to the variable declared in function.

- When the argument passed to a function is the value of a variable, the name of that variable is irrelevant to the function.

```python
def Sum(a, b):
    total = a + b
    print(total)

def main():
    x = 5
    y = 10
    Sum(x, y)
```

- The **value** of `x` and `y` were put into `a` and `b` respectively via function call.

- `x` and `y` are called: local variables to function main.

- `a` and `b` are called: local variables to function Sum.
Function: Arguments

- The name of argument passed to functions may or may not match the name of the variable used in the function.

```python
def Sum(a, b):
    total = a + b
    print(total)

def main():
    a = 5
    b = 10
    Sum(a, b)
```

- The value of `a` and `b` that are local to function `main` were put into the local variables `a` and `b` respectively via function call.
Functions: Returned Values

- Functions may return values (example: the result of a computation).

- Returned values can be:
  1. Printed
  2. Used in assignment statement
  3. Used in expression

```python
def Average(a, b):
    return (a+b)/2

def main():
    print (Average(10,2))
    avg = Average(3, 4)
    Total = Average(4,3) * 0.95
```
Functions with Multiple Returned Values

- Functions in Python may return multiple values

```python
def getabc():
    a = "Hello"
    b = "World"
    c = "!"
    return a, b, c

def main():
    a, b, c = getabc()
```
Example

def Sum(a, b, c):
    return (a+b+c)

def Greet(name, GPA):
    print("Hello", name)
    print("You have a GPA of ", GPA)

def Div(a, b):
    return a/b

def Mul(a, b):
    return a*b

def main()
    x = 3
    y = 4
    z = 2
    myStr = "Mike"
    Total = Sum(x, y, z)
    print (Greet(myStr))
    Result = Sum(x, y, z) + Mul(a, b) - Div(y, z)
Functions that Modify Variables (1)

What is the output of the following program:

```python
def Bonus(grade):
    grade = grade + 10

def main():
    myGrade = 75
    print (myGrade)
    Bonus(myGrade)
    print (myGrade)
```

Output: 75
75

Why?
Functions that Modify Variables (1)

- **myGrade** is an argument passed to function Bonus.
- **myGrade** is a numeric data type, also called immutable, that is a function cannot modify its value. In this case, only the value of myGrade matters.
- The function call will put the value of **myGrade** into **grade**
- **grade** is only known ‘locally’ to the function Bonus
- If you want to export the value from function Bonus back to main, function Bonus **MUST** use a **return** statement
- Then you can use the function call in an assignment statement
Bonus function now returns a value

```python
def Bonus(grade):
    grade = grade + 10
    return grade

def main():
    myGrade = 75
    print(myGrade)
    Bonus(myGrade)
    myGrade = Bonus(myGrade)
    print(myGrade)
```

Output: 75
85
What is the output of the following program:

```python
def Bonus(gradeList):
    for i in range(len(gradeList)):
        gradeList[i] = gradeList[i] + 10

def main():
    myGrades = [75, 90, 80]
    print (myGrades)
    Bonus(myGrades)
    print (myGrades)
```

Output: [75, 90, 80]  
[85, 100, 90]

Why?
Functions that Modify Variables (2)

- **myGrades** is an argument passed to function Bonus.
- **myGrades** is list, in Python, lists are mutable, that is a function can modify its value.
- **gradesList** is only known ‘locally’ to the function Bonus.
- The function call will work on the actual contents of **myGrades** under the name **gradesList**
Functions with more than one return statement

```python
def Bonus(grade):
    grade = grade + 10
    return grade
    grade = grade + 10
    return grade

def main():
    myGrade = 75
    print(myGrade)
    Bonus(myGrade)
    myGrade = Bonus (myGrade)
    print(myGrade)
```

Output: 75
85

A function call terminates once a return statement is encountered.
What can go wrong?

• **If your parrot is dead, consider this:**
  - Did you use the exact same names (case, spelling)?
  - All the lines in the block must be indented, and *indented* the same amount.
  - Variables in the command area don’t exist in your functions, and variables in your functions don’t exist in the command area.
  - The computer can’t read your mind.
    - It will only do exactly what you tell it to do.
    - In fact, programs always “work,” but maybe not how you intended!
Final QUESTIONS???