



FUNCTIONS

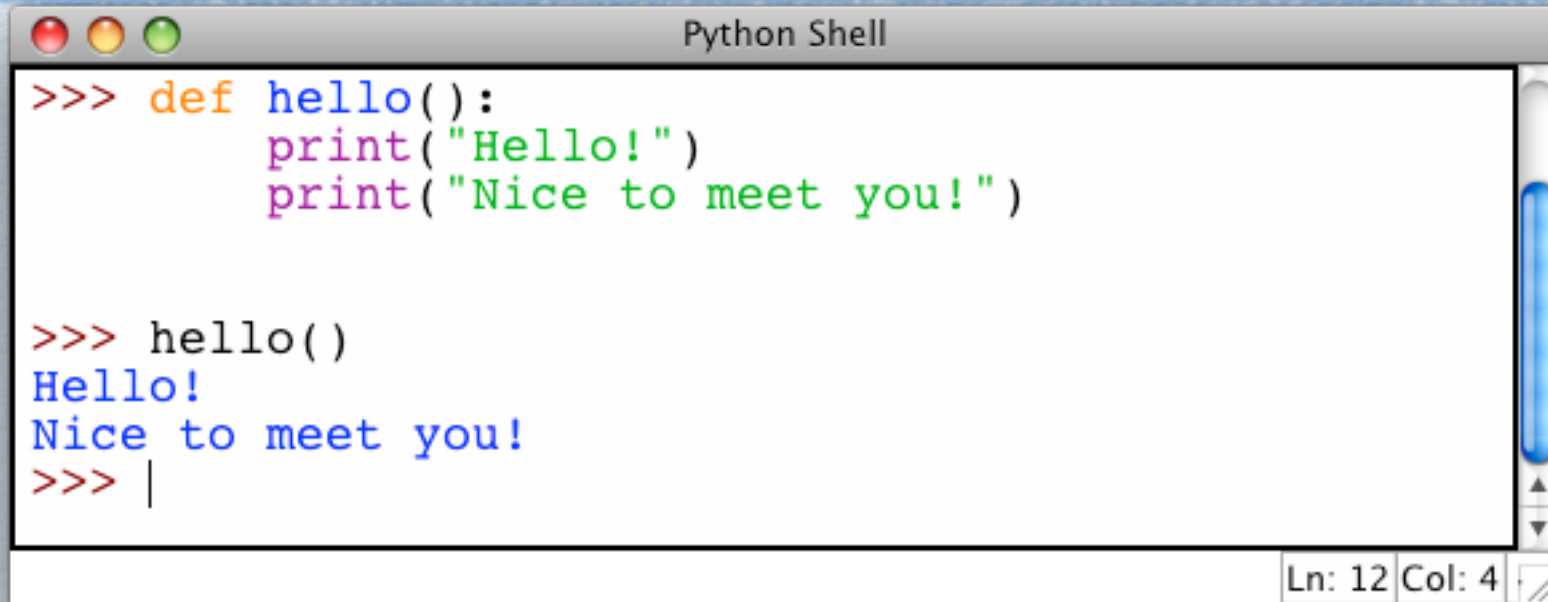
LEARNING OUTCOMES

- To understand why programmers divide programs up into sets of cooperating functions.
- To be able to define new functions in Python
- To understand the details of function calls and parameter passing in Python
- To write programs that use functions to reduce code duplication and increase program modularity.

FUNCTIONS

- A portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code.

YOU'VE SEEN THIS ALREADY



```
Python Shell
>>> def hello():
        print("Hello!")
        print("Nice to meet you!")

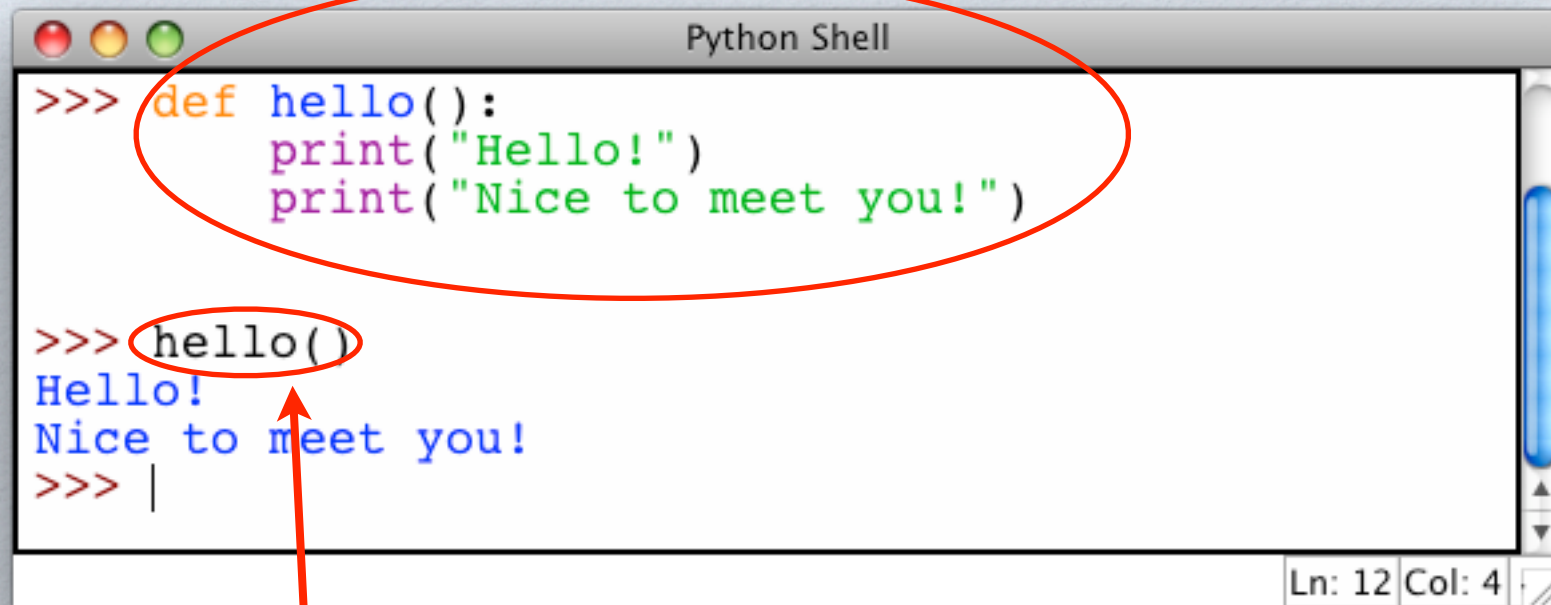
>>> hello()
Hello!
Nice to meet you!
>>> |
```

Ln: 12 Col: 4

- We have **defined** a function called `hello()`
- A **function** is a sequence of program statements with a name.
- Once we have defined the function, we can execute the programming statements anytime by referring to the function name.

DEFINING AND INVOKING FUNCTIONS

This is the **function definition**.
It creates the function.



```
Python Shell
>>> def hello():
    print("Hello!")
    print("Nice to meet you!")

>>> hello()
Hello!
Nice to meet you!
>>> |
```

The screenshot shows a Python Shell window with a title bar containing three colored buttons (red, yellow, green) and the text "Python Shell". The main area contains Python code. The first part is a function definition: `>>> def hello():` followed by two indented lines: `print("Hello!")` and `print("Nice to meet you!")`. This entire block is circled in red. Below it is the function invocation: `>>> hello()`, which is also circled in red. An arrow points from the `hello()` call to the `def hello():` definition. The output of the invocation is `Hello!` and `Nice to meet you!` on two separate lines. At the bottom right of the window, there is a status bar showing "Ln: 12 Col: 4".

We are **calling**, or **invoking**, the function here.
In other words, we are using the function.
Once a function has been defined, we can call it
whenever we like.

DEFINING AND CALLING FUNCTIONS

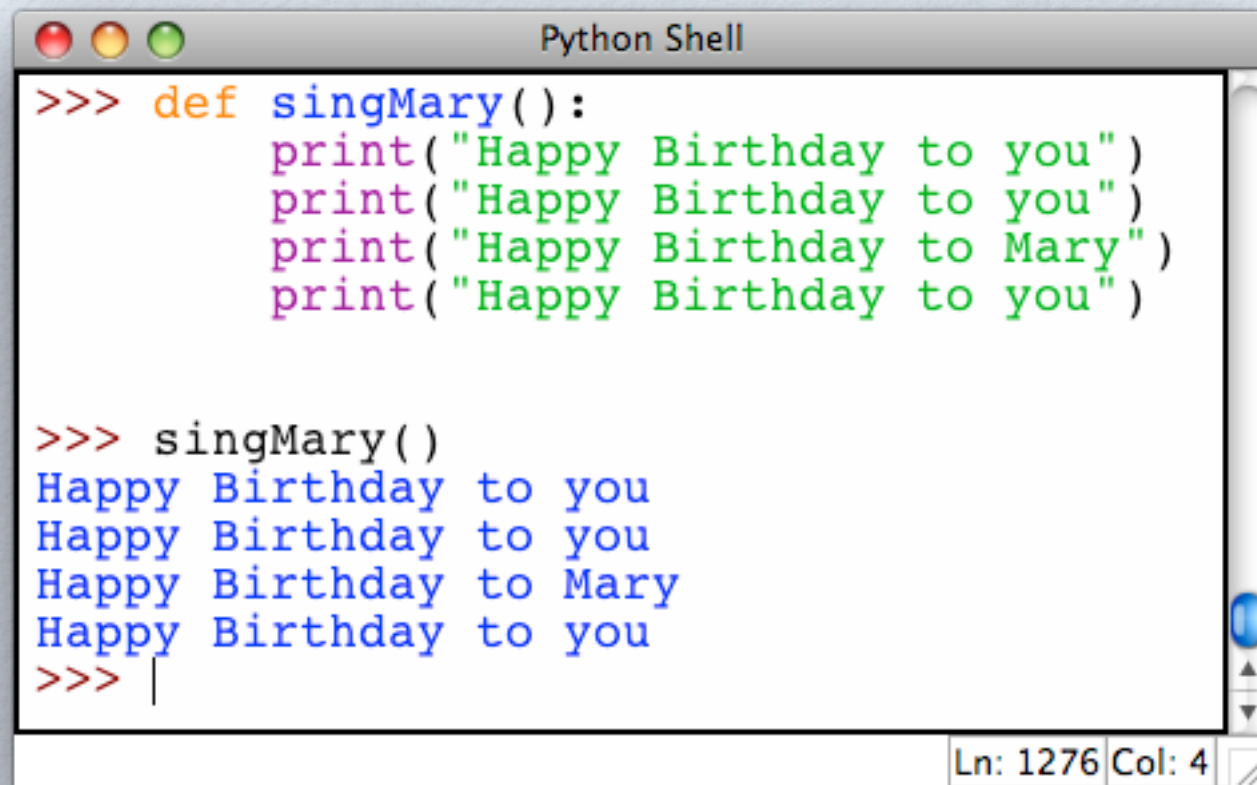
- A function definition looks like this:

```
def <name> ( <formal-parameters> ) :  
    <body>
```

- A function is called using its name, followed by a list of *actual parameter values*, or *arguments*.

```
<name> ( <actual-parameter-values> )
```

ANOTHER EXAMPLE

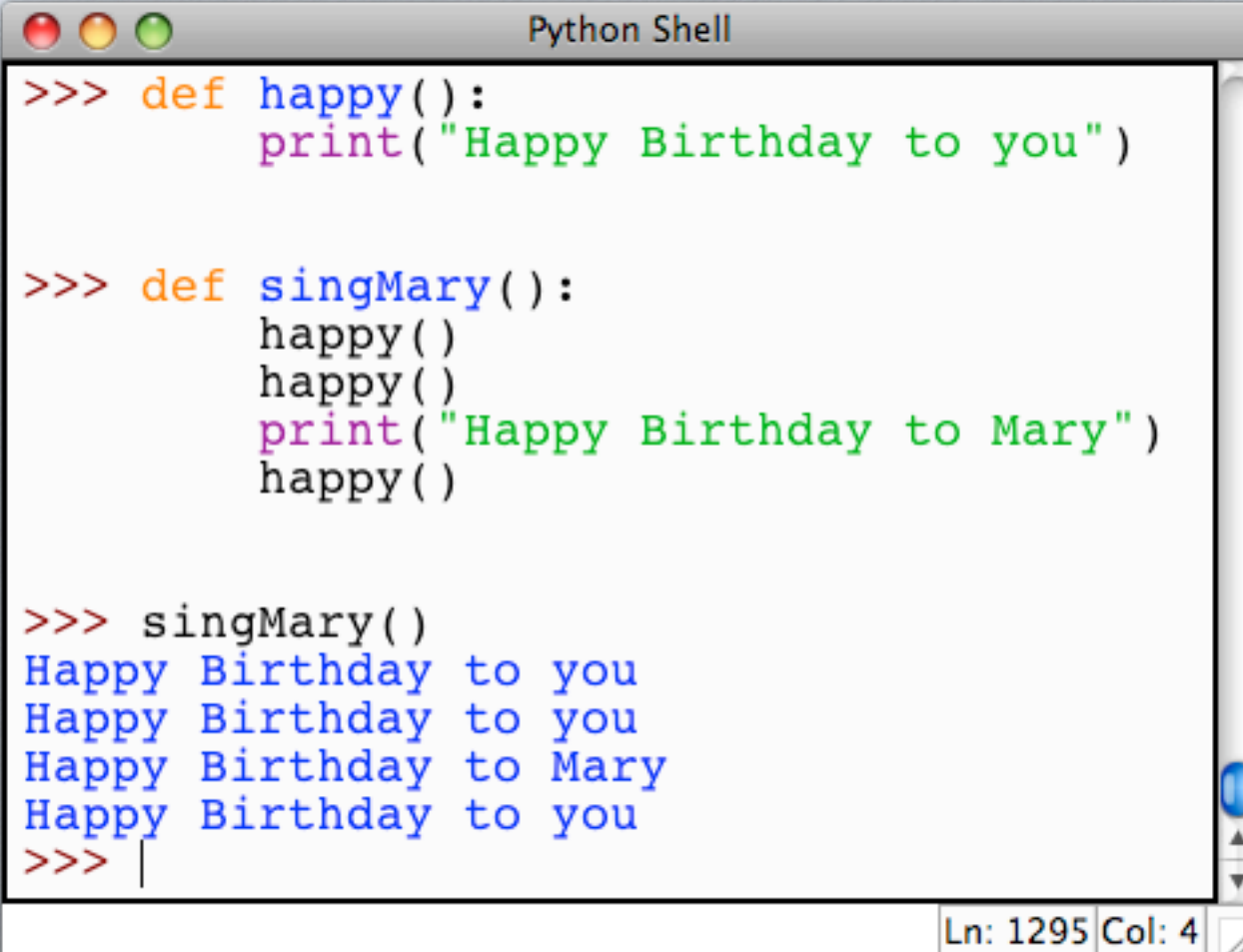


```
Python Shell
>>> def singMary():
    print("Happy Birthday to you")
    print("Happy Birthday to you")
    print("Happy Birthday to Mary")
    print("Happy Birthday to you")

>>> singMary()
Happy Birthday to you
Happy Birthday to you
Happy Birthday to Mary
Happy Birthday to you
>>> |
```

Ln: 1276 Col: 4

FUNCTIONS CAN CALL OTHER FUNCTIONS



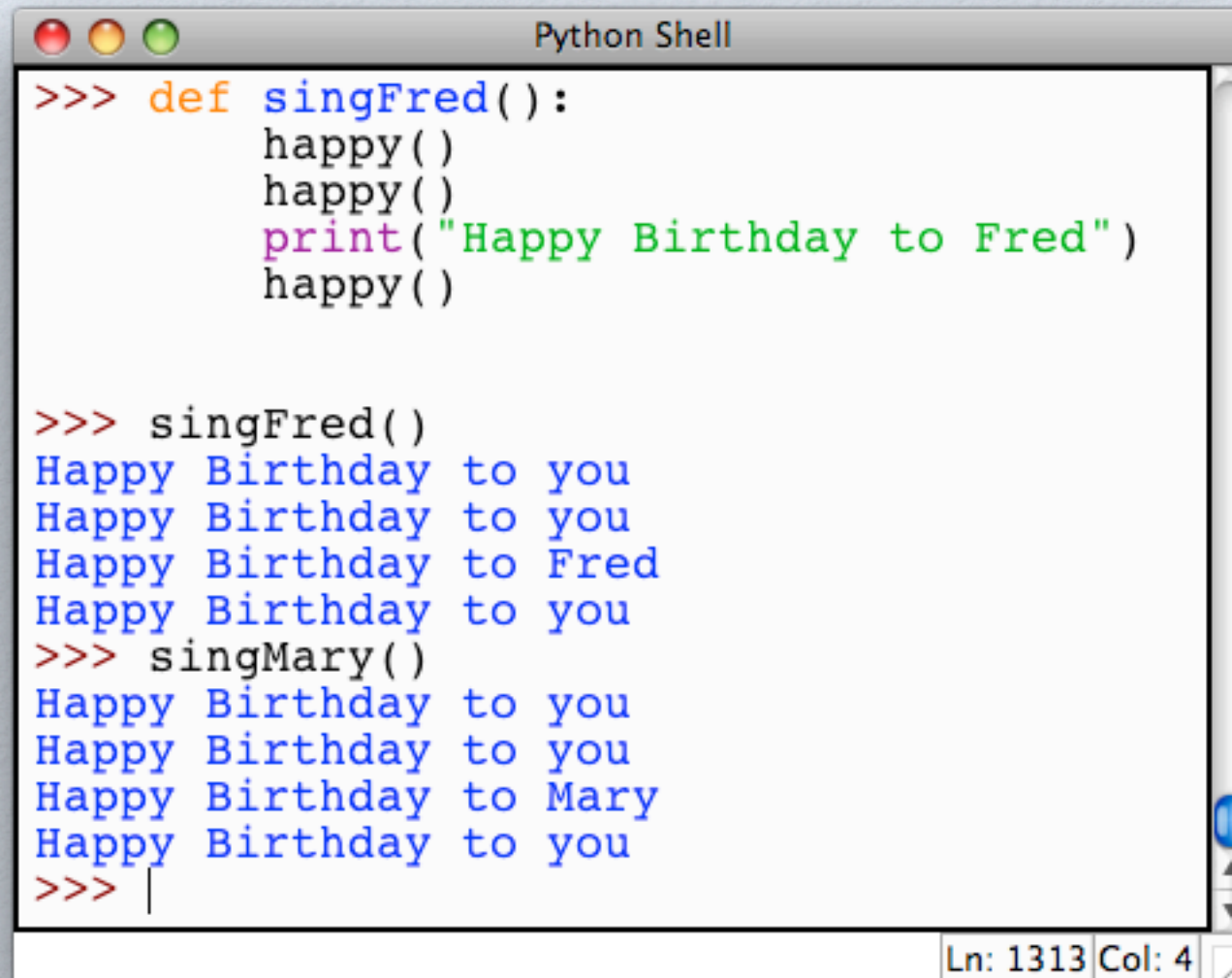
```
>>> def happy():
        print("Happy Birthday to you")

>>> def singMary():
        happy()
        happy()
        print("Happy Birthday to Mary")
        happy()

>>> singMary()
Happy Birthday to you
Happy Birthday to you
Happy Birthday to Mary
Happy Birthday to you
>>> |
```

Ln: 1295 Col: 4

DEFINED FUNCTIONS CAN BE CALLED BY MORE THAN ONE FUNCTION

A screenshot of a Python Shell window titled "Python Shell". The window contains Python code defining a function and calling it. The code is as follows:

```
>>> def singFred():  
    happy()  
    happy()  
    print("Happy Birthday to Fred")  
    happy()  
  
>>> singFred()  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday to Fred  
Happy Birthday to you  
>>> singMary()  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday to Mary  
Happy Birthday to you  
>>> |
```

The output shows that the `singFred()` function calls `happy()` three times and prints "Happy Birthday to Fred". The `singMary()` function calls `happy()` three times and prints "Happy Birthday to Mary". The status bar at the bottom right of the window shows "Ln: 1313 Col: 4".

FUNCTIONS WITH PARAMETERS

```
Python Shell
>>> def sing(person):
        happy()
        happy()
        print("Happy Birthday to", person)
        happy()

>>> sing("Mary")
Happy Birthday to you
Happy Birthday to you
Happy Birthday to Mary
Happy Birthday to you
>>> sing("Fred")
Happy Birthday to you
Happy Birthday to you
Happy Birthday to Fred
Happy Birthday to you
>>> |
```

Ln: 1345 Col: 4

- Parameters are customizable parts of a function
- They are variables that are **initialized** when the function is executed.
- We provide the value of the parameter when we call the function.

WHEN A FUNCTION IS CALLED

- When we call a function, a five-step process happens:
- Step 1: The calling function suspends execution at the point of the call.
- Step 2: The function starts running. Any formal parameter variables are created.
- Step 3: The formal parameters variables are initialized with the values supplied by the actual parameter values in the call.
- Step 4: The body of the function is executed.
- Step 5: Program flow continues at the point just after the function was called.

EXAMPLE

```
Python Shell
>>> def sing(person):
        happy()
        happy()
        print("Happy Birthday to", person)
        happy()

>>> sing("Mary")
Happy Birthday to you
Happy Birthday to you
Happy Birthday to Mary
Happy Birthday to you
>>> sing("Fred")
Happy Birthday to you
Happy Birthday to you
Happy Birthday to Fred
Happy Birthday to you
>>> |
```

Ln: 1345 Col: 4

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```


```
sing("Mary")  
sing("Fred")
```



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```



```
sing("Mary")  
sing("Fred")
```



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

"Mary"

```
→ sing("Mary")  
   sing("Fred")
```



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person   
    happy()  
    print("Happy birthday to", person)  
    happy()
```

"Mary"

```
→ sing("Mary")  
   sing("Fred")
```



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

"Mary"



```
sing("Mary")  
sing("Fred")
```



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

```
→ sing("Mary")  
   sing("Fred")
```

"Mary"



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

```
→ sing("Mary")  
   sing("Fred")
```

"Mary"

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

```
→ sing("Mary")  
   sing("Fred")
```



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

```
→ sing("Mary")  
   sing("Fred")
```

Happy birthday to you

TRACING THE FLOWS

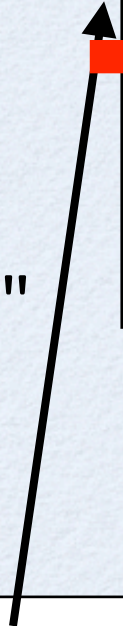
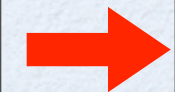
```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

Happy birthday to you

```
→ sing("Mary")  
   sing("Fred")
```

"Mary"



TRACING THE FLOWS

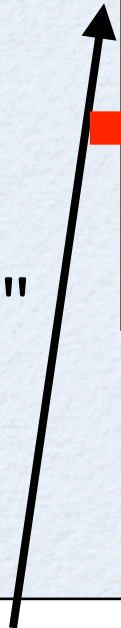
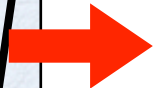
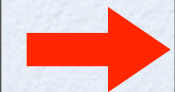
```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()            
    print("Happy birthday to", person)  
    happy()
```

Happy birthday to you

```
→ sing("Mary")  
   sing("Fred")
```

"Mary"



TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

Happy birthday to you

```
→ sing("Mary")  
   sing("Fred")
```

"Mary"

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

```
→ sing("Mary")  
   sing("Fred")
```

Happy birthday to you

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

```
→ sing("Mary")  
   sing("Fred")
```

```
Happy birthday to you  
Happy birthday to you
```

TRACING THE FLOWS

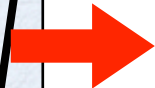
```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()            
    print("Happy birthday to", person)  
    happy()
```

```
Happy birthday to you  
Happy birthday to you
```

```
→ sing("Mary")  
   sing("Fred")
```

"Mary"



"Mary"

TRACING THE FLOWS

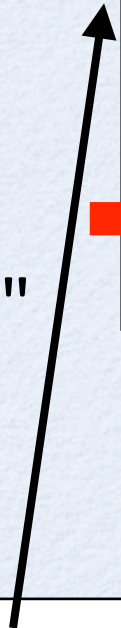
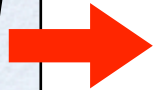
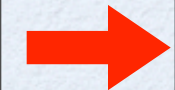
```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

```
Happy birthday to you  
Happy birthday to you
```

```
→ sing("Mary")  
   sing("Fred")
```

"Mary"



"Mary"

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

"Mary"

```
→ sing("Mary")  
   sing("Fred")
```

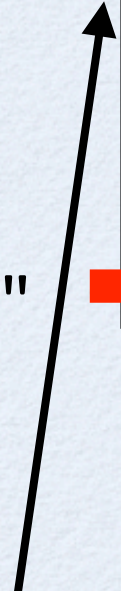
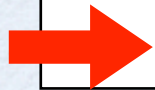
```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary
```

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

"Mary"



```
sing("Mary")  
sing("Fred")
```

```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary
```

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary
```

```
→ sing("Mary")  
   sing("Fred")
```

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

```
→ sing("Mary")  
   sing("Fred")
```

```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary
```

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

person "Mary"

```
→ sing("Mary")  
   sing("Fred")
```

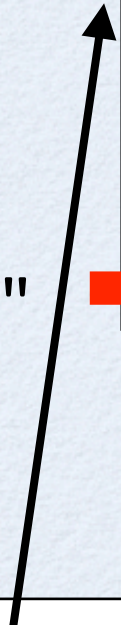
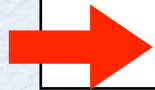
```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary  
Happy birthday to you
```

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()           person "Mary"  
    happy()  
    print("Happy birthday to", person)  
    happy()
```

"Mary"




```
sing("Mary")  
sing("Fred")
```

```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary  
Happy birthday to you
```

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```




```
sing("Mary")  
sing("Fred")
```

```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary  
Happy birthday to you
```

TRACING THE FLOWS

```
def happy():  
    print("Happy birthday to you")
```

```
def sing(person):  
    happy()  
    happy()  
    print("Happy birthday to", person)  
    happy()
```



```
sing("Mary")  
sing("Fred")
```

```
Happy birthday to you  
Happy birthday to you  
Happy birthday to Mary  
Happy birthday to you
```

EXERCISE

- What are the outputs of the programs below?

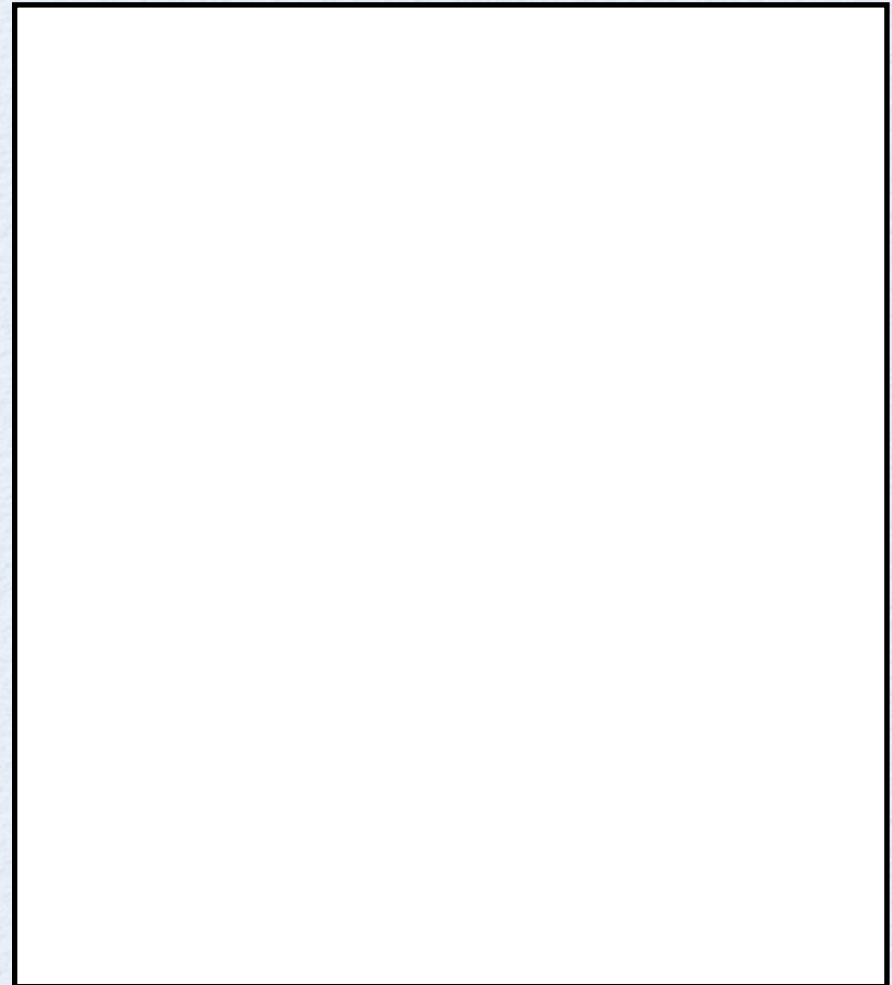
```
exercise1.py
exercise1.py (~/comp201/Lectures/07 Function...
1 def myFunction():
2     print("-", end="")
3
4 def main():
5     print("A", end="")
6     myFunction()
7     print("B", end="")
8     myFunction()
9     print("C", end="")
10    myFunction()
11
12 main()
1:12 (155, MacRoman) - - - UG 9/21Mb 1:30 AM
```

```
exercise2.py
exercise2.py (~/comp201/Lectures/07 Function...
1 def myFunction(c):
2     print(c, end="")
3
4 def main():
5     c = "A"
6     myFunction(c)
7     c = "B"
8     myFunction(c)
9     c = "C"
10    myFunction(c)
11
12 main()
1:17 (161, MacRoman) - - - UG 9/21Mb 1:33 AM
```

EXERCISE

- What is the output of the program below?

```
def grunt():  
    print "growl"  
  
def squark():  
    print "squeak"  
    print "honk"  
  
def main():  
    print "howl"  
    squark()  
    print "beep"  
    grunt()  
    squark()  
  
main()
```



EXERCISE

- Fill in the blanks below with the following words (one word per blank):

sequentially functions block indented executed

- Program statements are _____ in a defined order. Execution starts with the first non-_____ statement in the program which is not a function definition. Execution continues _____ through each program _____ jumping into _____ when these are called.

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```

```
>>> calculateInterest(150, 3)
```

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```

amount

```
>>> calculateInterest(150, 3)
```

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```

amount

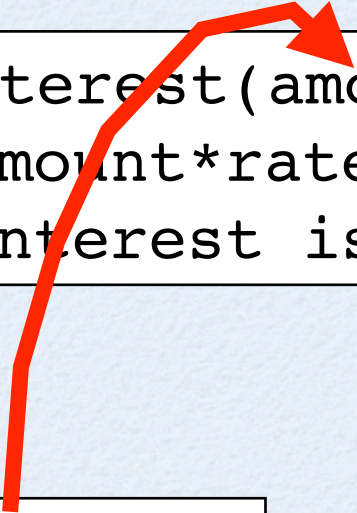
rate

```
>>> calculateInterest(150, 3)
```

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```



amount

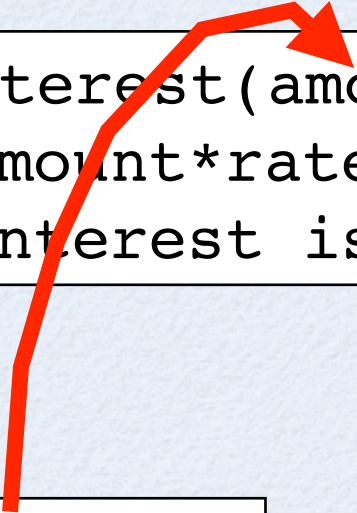
rate

```
>>> calculateInterest(150, 3)
```

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```



amount

150

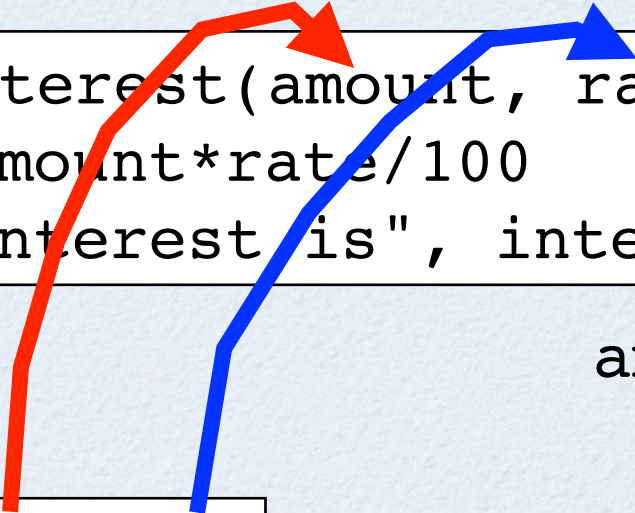
rate

```
>>> calculateInterest(150, 3)
```

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```



amount

150

rate

```
>>> calculateInterest(150, 3)
```

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```

```
>>> calculateInterest(150, 3)
```

amount

150

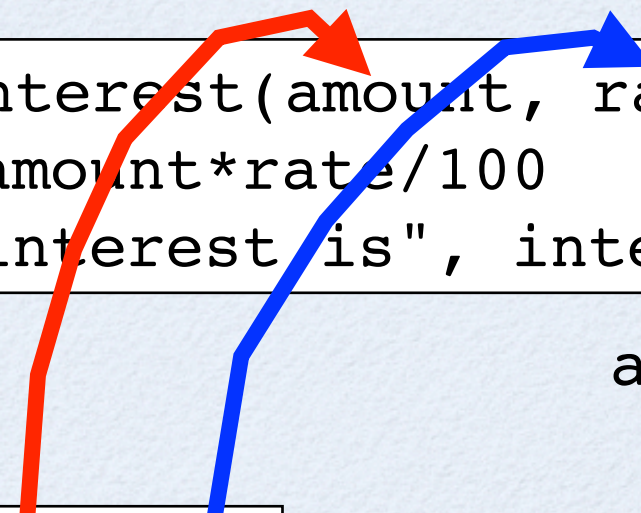
rate

3

MULTIPLE PARAMETERS

- A function can have more than one parameter.
- When a function definition has multiple parameters, the actual parameter in a function call are matched up with the formal parameters **by position**.

```
def calculateInterest(amount, rate):  
    interest = amount*rate/100  
    print("The interest is", interest)
```



amount

150

rate

3

```
>>> calculateInterest(150, 3)  
The interest is 4.5
```

FUNCTIONS RETURNING VALUES

- We have seen the following before:

```
x = math.sqrt(4)    # x will get the value 2
```

```
y = len("Hello")   # y will get the value 5
```

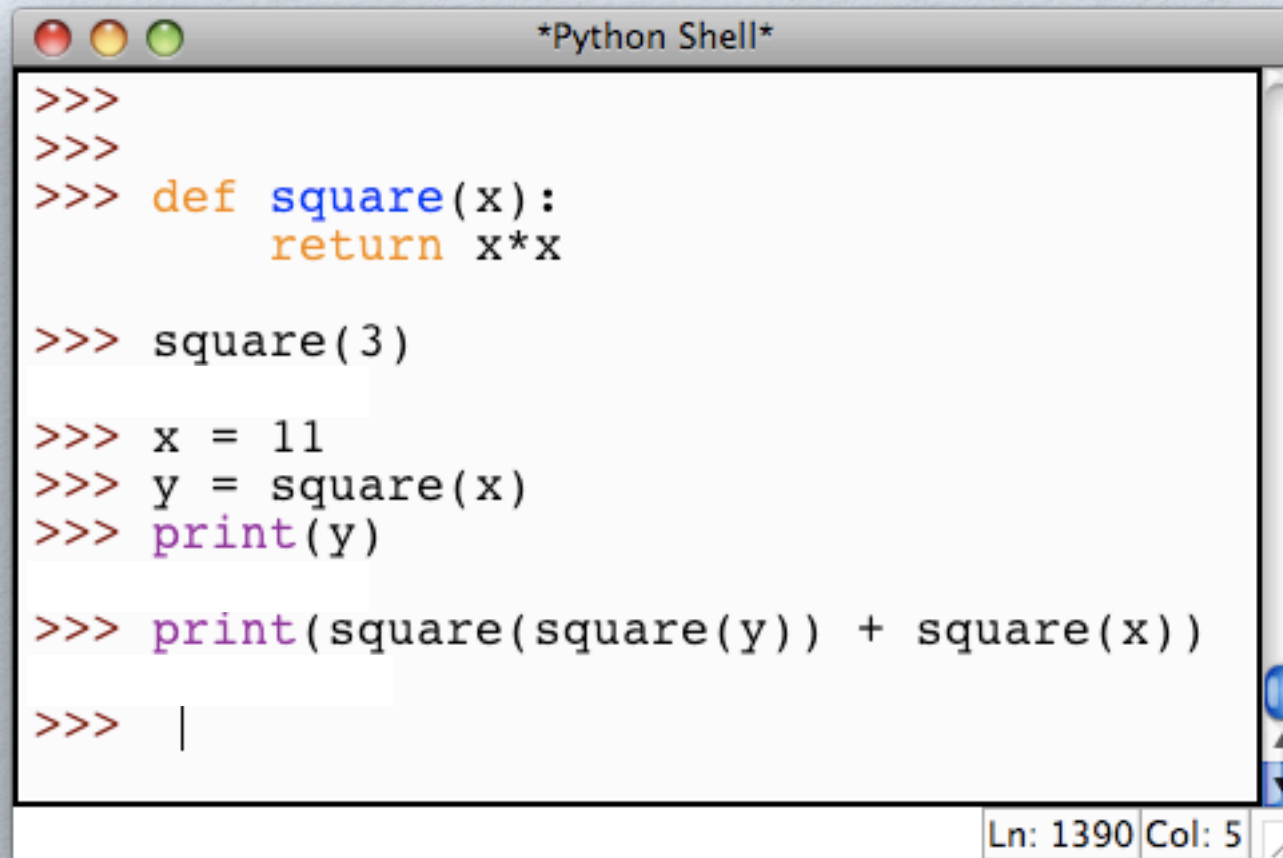
```
distance = milesToKm(20)
```

- Anything that has a pair of parentheses `(())` at the end is a function in Python.
- So these functions are giving “answers”!

FUNCTIONS RETURNING ANSWERS

- To make a function return an answer, we use the **return** keyword, and put the value that we want to use as the function's answer after the return.
- The value of the function will then be equals to its answer.

FILL IN THE OUTPUT



```
*Python Shell*
>>>
>>>
>>> def square(x):
>>>     return x*x

>>> square(3)

>>> x = 11
>>> y = square(x)
>>> print(y)

>>> print(square(square(y)) + square(x))

>>> |
```

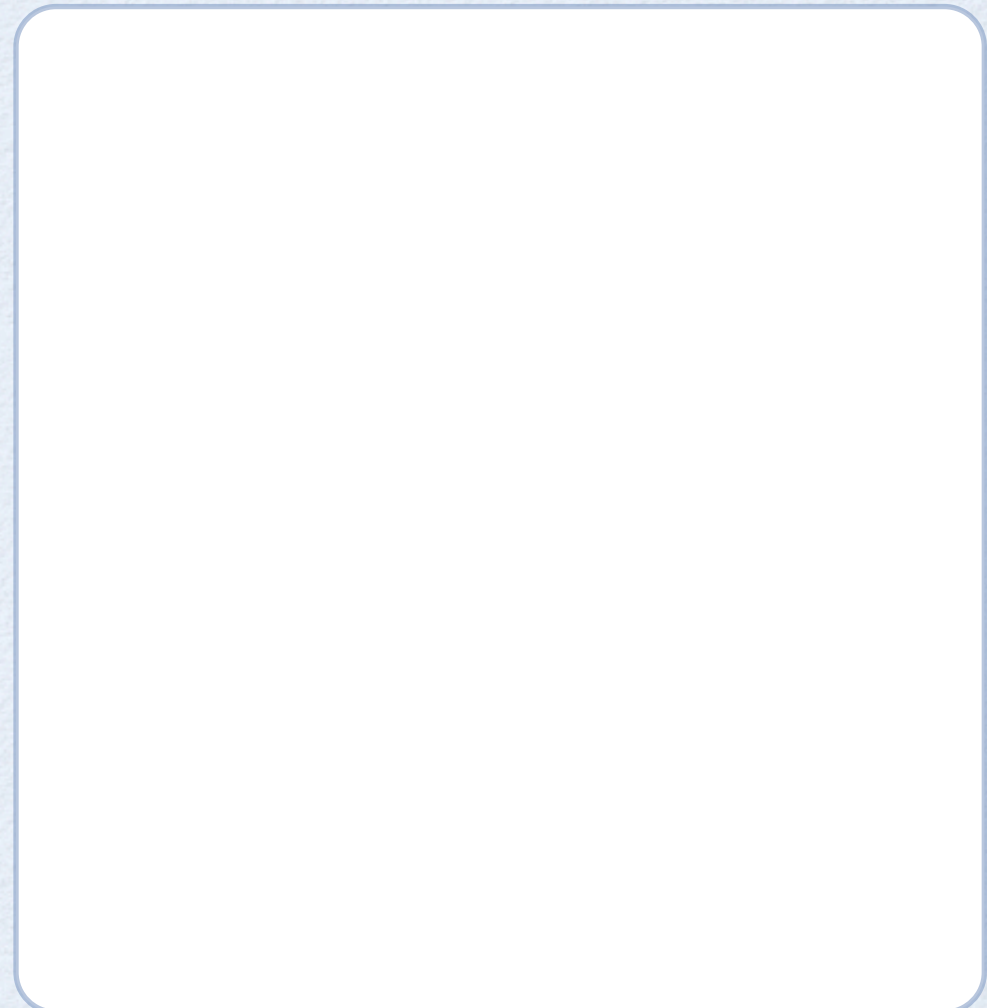
Ln: 1390 Col: 5

- Can you trace the flow of the last statement?

EXERCISE

- What is the output of this program?

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x  
  
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)  
  
main()
```



LOCAL VARIABLES

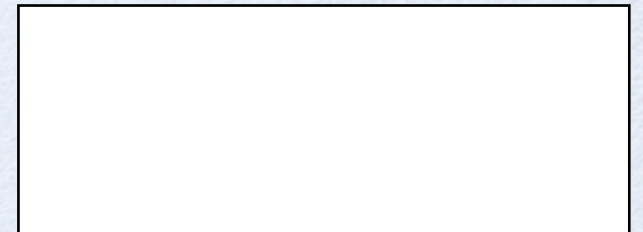
- Note how both `main()` and `myFunction()` both had a variable called `x`.
- However, a change to `x` in `myFunction()` did not affect the value of `x` in `main()`.
- That's because there are two variables, one belonging to `myFunction()`, and one belonging to `main()`.
- Both are called `x` but they are independent.

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

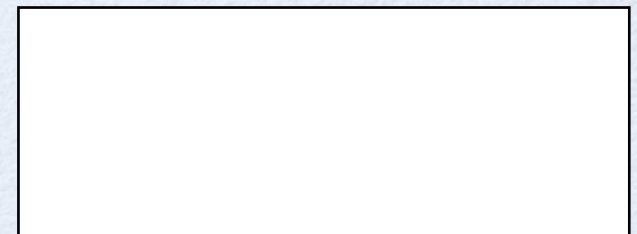
main()



TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```



 main()

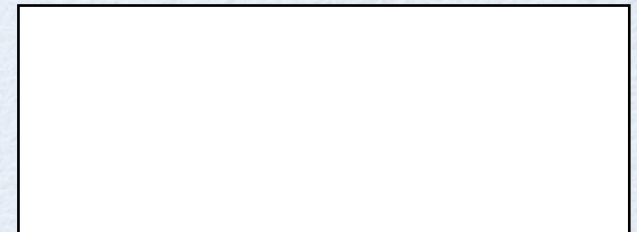
TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```



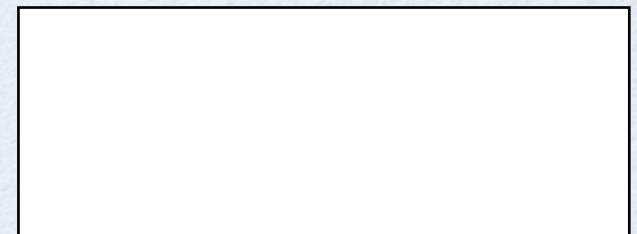
main()



TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```



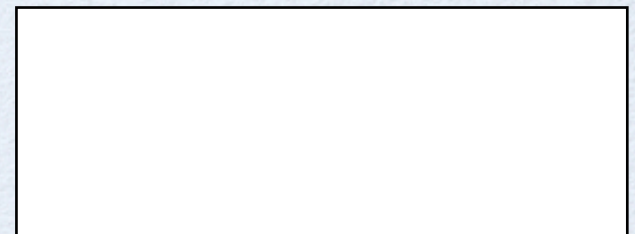
main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```



 main()

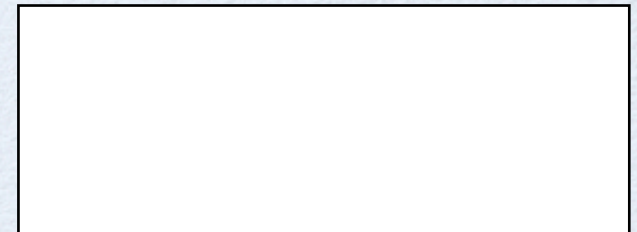
TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

 main()



TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

In Main: 1

 main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

In Main: 1

 main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

In Main: 1

 main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

x 1

In Main: 1

 main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

x 1

In Main: 1

main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

x 2

In Main: 1

main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

x 2

In Main: 1

main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):
```

```
    x = x+1
```

```
    print("In Function:", x)  
    return x
```

x 2

In Main: 1

In Function: 2

main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

x 2

```
In Main: 1  
In Function: 2
```

main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

```
In Main: 1  
In Function: 2
```

 main()

2

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

```
In Main: 1  
In Function: 2
```

 main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

x 1

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

```
In Main: 1  
In Function: 2  
In Main: 1
```

 main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

```
In Main: 1  
In Function: 2  
In Main: 1
```



main()

TRACING THE FLOWS

```
def main():  
    x = 1  
    print("In Main:", x)  
    myFunction(x)  
    print("In Main:", x)
```

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x
```

main()

```
In Main: 1  
In Function: 2  
In Main: 1
```

EXERCISE

- What is the output of this program?

```
def myFunction(x):  
    x = x+1  
    print("In Function:", x)  
    return x  
  
def main():  
    x = 1  
    print("In Main:", x)  
    x = myFunction(x)  
    print("In Main:", x)  
    x = myFunction(x)  
    print("In Main:", x)  
  
main()
```



EXERCISE

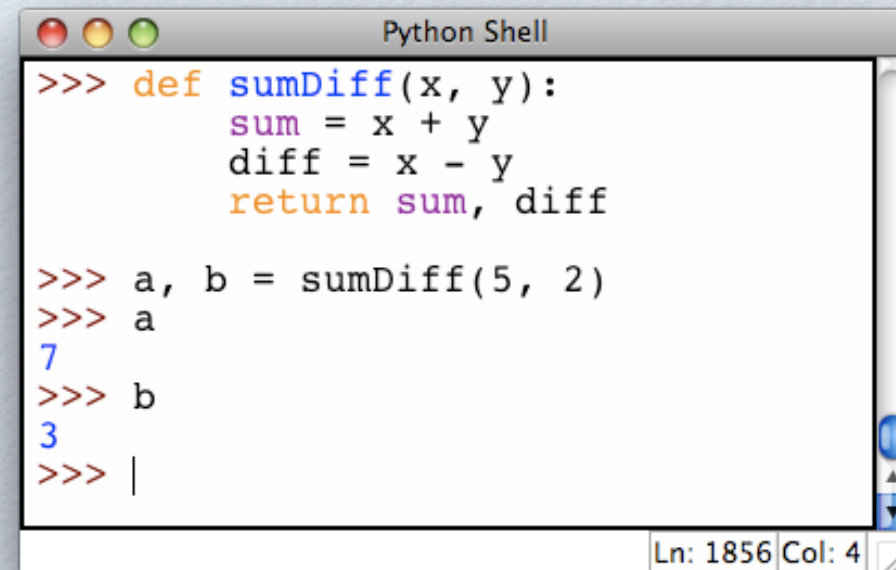
- Write a function, `countDown()`, that will work with the given `main()` function to generate *exactly* the given output.
- Hint: use a definite loop with a range, and `print()` with `end=" "`

```
def main():  
    count = eval(input("Give me a whole number: "))  
    countDown(count)  
  
main()
```

```
C:\> python countDown.py  
Give me a whole number: 10  
10 9 8 7 6 5 4 3 2 1 Go!  
C:\>
```

MULTIPLE RETURN VALUES

- Unlike some other programming languages (e.g. C, Java), Python allows functions to return multiple values.
- A function with multiple return values should be placed into a simultaneous assignment program statement.
- As with parameters, when multiple values are returned, they are assigned to the variables by position



```
Python Shell
>>> def sumDiff(x, y):
      sum = x + y
      diff = x - y
      return sum, diff

>>> a, b = sumDiff(5, 2)
>>> a
7
>>> b
3
>>> |
```

Ln: 1856 Col: 4

OBJECTS AS PARAMETERS

- So far, we have been passing simple values (such as numbers) into the functions as arguments.
- However, *anything* can be passed into a function as an argument.
- This includes strings, objects, lists, etc.

PASSING BY VALUES

```
def addInterest(amount, rate):  
    interest = amount*rate/100  
    amount = amount + interest  
  
def main():  
    amount = 100  
    rate = 3  
    print("1. Amt:", amount)  
    addInterest(amount, rate)  
    print("2. Amt:", amount)  
  
main()
```

- From previous slides, we know that in this program:
 - The variable amount in addInterest() and the variable amount in main() are two different variables.
 - A change to amount in addInterest() will not change the value of amount in main()

PASSING BY VALUES

```
def addInterest(amount, rate):  
    interest = amount*rate/100  
    amount = amount + interest  
  
def main():  
    amount = 100  
    rate = 3  
    print("1. Amt:", amount)  
    addInterest(amount, rate)  
    print("2. Amt:", amount)  
  
main()
```

- What is happening here is:
 - The **value** of the variable `amount` gets passed to the function `addInterest()`.
 - The function makes its own variable called `amount`, and copies the passed value into the new variable.
 - As this is a new variable, any changes do not affect `main()`'s `amount`.
- This is called **passing by value**.

HOW ABOUT THIS PROGRAM?

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        interest = amounts[i]*rate/100  
        amounts[i] = amounts[i] + interest
```

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

```
main()
```

```
J:\> python CalculateInterests.py  
Before: [1000, 2000, 3500, 4800]  
After: [1030.0, 2060.0, 3605.0, 4944.0]  
J:\>
```

FUNCTIONS AND LISTS

- Unlike the previous example, it seems that the function is able to modify the contents of the `amounts` list!
- But didn't we just say that parameters are passed by value? Then how can the function `addInterests()` make changes to the list in `main()`?

FUNCTIONS AND LISTS

- Recall that variables in Python are only able to store one number at a time.
- So the variable `amounts` in `main()` actually stores the **reference** of the list.
- When the function `addInterests()` is called, what gets passed is the value of the variable `amounts` -- which is the **reference** of the list.


FUNCTIONS AND LISTS

main()

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```


FUNCTIONS AND LISTS

 main()

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS



main()

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

```
def main():  
→ amounts = [1000, 2000, 3500, 4800]  
  rate = 3  
  print("Before:", amounts)  
  addInterests(amounts, rate)  
  print("After:", amounts)
```

```
def addInterests(amounts, rate):  
  for i in range(len(amounts)):  
    amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

```
def main():  
→ amounts = [1000, 2000, 3500, 4800]  
  rate = 3  
  print("Before:", amounts)  
  addInterests(amounts, rate)  
  print("After:", amounts)
```

1000	2000	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
  for i in range(len(amounts)):  
    amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

```
def main():  
→ amounts = [1000, 2000, 3500, 4800]  
  rate = 3  
  print("Before:", amounts)  
  addInterests(amounts, rate)  
  print("After:", amounts)
```

amounts

1000	2000	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
  for i in range(len(amounts)):  
    amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    → rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

1000	2000	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

main()

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1000	2000	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1000	2000	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1000	2000	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1000	2000	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1000 | 2000 | 3500 | 4800

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

amounts

rate

3

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1000 | 2000 | 3500 | 4800

amounts

```
→ def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

rate

3

i

0

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030 | 2000 | 3500 | 4800

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
    → amounts[i] = amounts[i]*(1+rate/100)
```

amounts

rate

3

i

0

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030 | 2000 | 3500 | 4800

amounts

```
→ def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

rate

3

i

1

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030	2060	3500	4800
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
    → amounts[i] = amounts[i]*(1+rate/100)
```

amounts

rate

i

3

1

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030 | 2060 | 3500 | 4800

amounts

```
→ def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

rate

3

i

2

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030 | 2060 | 3605 | 4800

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
    → amounts[i] = amounts[i]*(1+rate/100)
```

amounts

rate

i

3

2

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030 | 2060 | 3605 | 4800

amounts

rate

i

3

3

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030 | 2060 | 3605 | 4944

amounts

rate

3

i

3

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
    → amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    → addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030	2060	3605	4944
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

FUNCTIONS AND LISTS

→ main()

Before: [1000, 2000, 3500, 4800]
After: [1030, 2060, 3605, 4944]

```
def main():  
    amounts = [1000, 2000, 3500, 4800]  
    rate = 3  
    print("Before:", amounts)  
    addInterests(amounts, rate)  
    print("After:", amounts)
```

amounts

rate

3

1030	2060	3605	4944
------	------	------	------

```
def addInterests(amounts, rate):  
    for i in range(len(amounts)):  
        amounts[i] = amounts[i]*(1+rate/100)
```

PASSING PARAMETERS

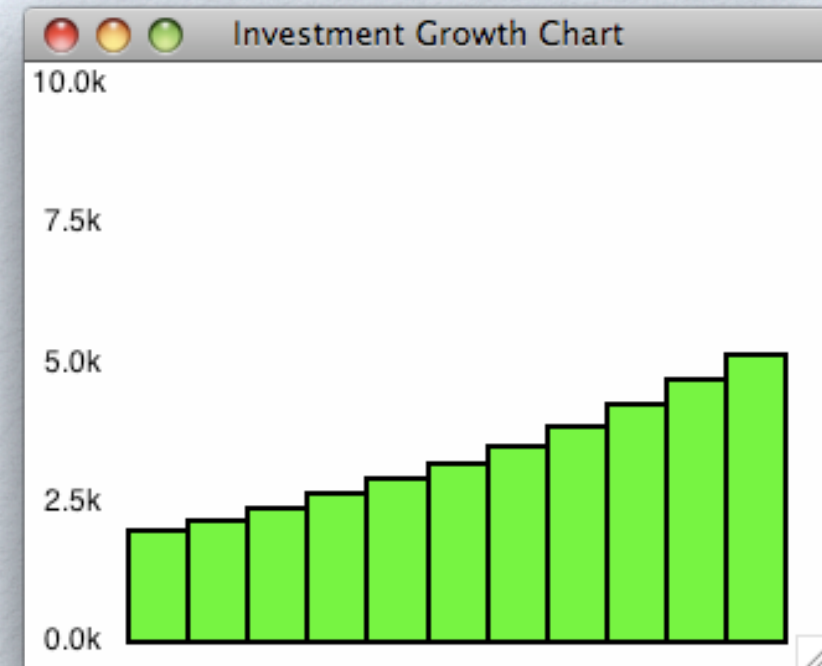
- In Summary:
 - All parameters are passed by value (i.e. the value of the parameter variable gets sent over to the function)
 - If the parameter is a number, then changes to the parameter variable in the function will not affect the variable in `main ()` (or the calling function).
 - However, if the parameter is a reference to a mutable object (such as a list or a graphics object), then changes to the state of the object will be visible in `main ()`.

WHY ARE FUNCTIONS USEFUL?

- Functions are useful for several reasons:
 - They (usually) make the program shorter and save effort on the part of the programmer.
 - They reduce the possibility of errors.
 - Properly named and used, they make the program more readable.

EXAMPLE

- Consider the following output and the program code (on the following page) that generates it.
- Without seeing the rest of the program, what do the functions `createLabeledWindow()` and `drawBar()` do?
- What are the steps taken by the program?
- Supposing we want to change the y-axis labels to one label per 2k, rather than 2.5k. Which function would you need to change?
- How do we know?



EXAMPLE

```
def main():
    principal = eval(input("How much did you invest?"))
    rate = eval(input("What is the interest per annum?"))

    window = createLabeledWindow()
    drawBar(window, 0, principal)
    for year in range(1, 11):
        principal = principal * (1+interest/100)
        drawBar(window, year, principal)

    input("Press <enter> to quit")
    win.close()
```

SUMMARY

- A properly designed program will use functions and comments to make itself **readable** and **maintainable**.
- With properly named and designed functions, the program can go a long way towards being **self-documenting**.
- We will talk more about program design later in the semester.
- For now, focus on using functions to make your code **modular** by separating repeated statements into functions.

YOU NOW KNOW...

- Why programmers divide programs up into sets of cooperating functions.
- How to define new functions in Python
- The details of function calls and parameter passing in Python
- How to write programs that use functions to reduce code duplication and increase program modularity.