



CS 1112: Introduction To Programming

Functions and Scope

Dr. Nada Basit // `basit[at]Virginia[dot]edu`

Friendly Reminders

- Your **safety** and **comfort** is important!
 - If you choose to wear a mask you are welcome to do so
 - *We will interpret wearing a mask as being considerate and caring of others in the classroom (not that you are sick), and realize that some may choose to mask to remain distanced*
- Remember to always be **kind, respectful, supportive, compassionate** and **mindful of others!** 😊
- Be an **active** participant in your learning!
You're welcome and **encouraged** to ask questions during class!
- If you feel **unwell**, or think you are, **please stay home**
 - *Contact us! We will work with you!*
 - Get some rest 😊
 - View the recorded lectures – *please allow 24-48 hours to post*



Announcements

- **Quiz 3** is being graded
- **PA02** is **due by 11:00pm on Wednesday (tonight)!**
 - Submit on Gradescope: your .py file and a reflection file (PDF).
 - *Not sure how to create/submit a PDF? No problem! Ask one of the TAs for help!*
- **PA03** (on Functions) is **due by 11:00pm on Feb 19!**
- Coming up...
 - **Exam 1** on February 26, 2025 (during class time)
 - If you have **SDAC accommodations**, please book an appointment at the SDAC facility to take the exam with your extended time and distraction-free environment.
 - You can book a testing time slot at any time on Feb. 26th but you must book on this day (not another day). Please contact SDAC today ☺



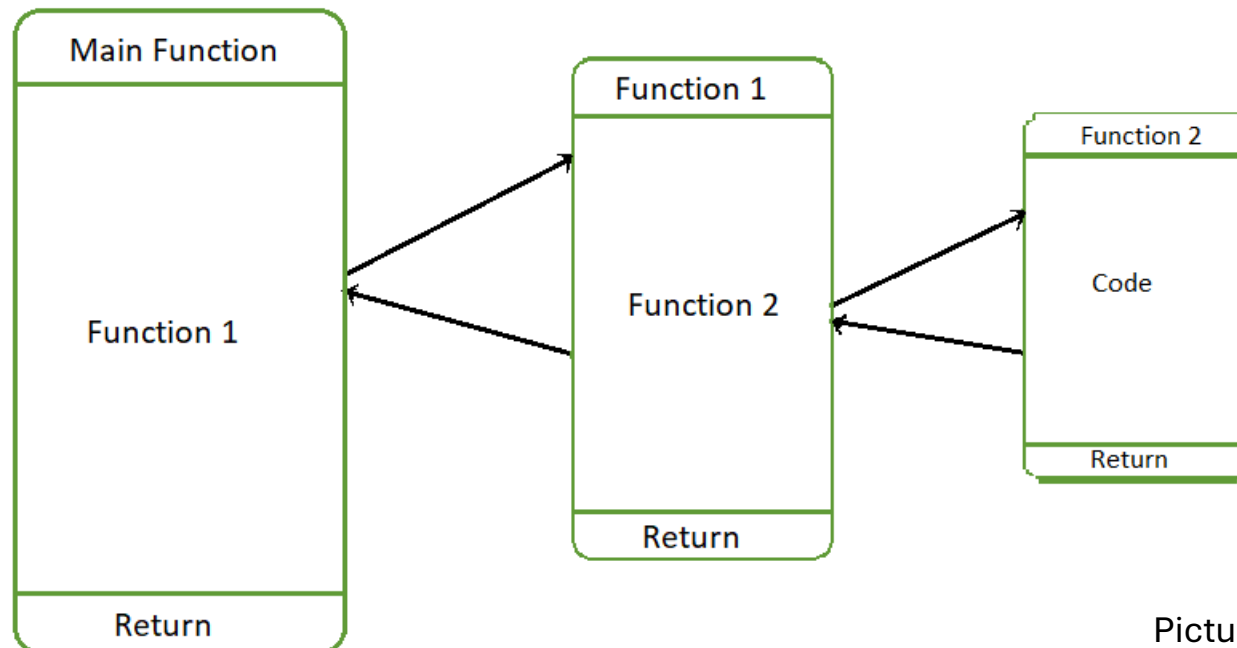
Functions calling Functions

Calling another function from inside a function

Visual: functions calling other functions...

In the below figure. The function call is made from the **Main function** to **Function1**. Now the state of the Main function is stored in Stack, and execution of the Main function is continued when Function 1 returns. When Function1 calls **Function2**, the state of the Function1 is stored in the stack, and execution of Function 1 will be continued when Function 2 returns.

Calling Function
execution will be
completed only when
Called Function's
execution is completed



Functions calling other Functions

- To call a function from another function in Python, you just have to **call the name of the function** and pass in any required arguments. See the following example:

```
def func1():  
    print("This is function 1")
```

```
def func2():  
    print("This is function 2")  
    func1() # call function 1
```

```
# Call function 2 –  
func2()
```

Output:

```
This is function 2  
This is function 1
```

In the above code, function 2 simply calls function 1 by **invoking the name of the function**. When you run the code, you'll see that function 2 executes first and then function 1 executes.

What happens when you invoke (call) a function?

1. Python **creates** *memory* for the function (*we don't see this*)
2. **Argument** values are assigned to parameter names
3. Lines of code in the body of the function are *executed*
4. When the function encounters a **return** statement, a value is *passed back* to the **caller**, and the function **ends**
5. Python **removes** the *memory* for the function (all values that existed *only in the function* are gone)



PYTHON DEMONSTRATION

Let's jump on PyCharm!

`functions_calling_functions.py`

This raises the issue of Scope...

When writing functions, you will often be dealing with a lot of variables. It is important to understand variable scope...

This raises the issue of Scope...

- OK... maybe not that kind of Scope... ☺



Scope

- The area where a variable is recognized (or has meaning)
- **Global Variables** - declared outside of a function.
 - Have scope from where they are first created until the end of the program (or module/file)
- **Local Variables** - declared inside of a function.
 - Have scope from where they are first created until the end of the function definition
 - This includes parameters

- In this example, how would you describe the scope of the variable x?

???

- What about variable b?

???

```
def add_stuff(a):  
    x = a + 5  
    return x  
  
b = 4  
print("Statement 1: The value of b is", b)  
y = add_stuff(7)  
print("Statement 2: The value of y is", y)  
print("Statement 3: The value of b is", b)
```

Note: Tools like pythontutor.com can be helpful in visualizing the concept of scope

Scope

- The area where a variable is recognized (or has meaning)
- **Global Variables** - declared outside of a function.
 - Have scope from where they are first created until the end of the program (or module/file)
- **Local Variables** - declared inside of a function.
 - Have scope from where they are first created until the end of the function definition
 - This includes parameters
- In this example, how would you describe the scope of the variable x?
 - a) **x is a local variable**
 - b) **It is local to function add_stuff**
- What about variable b?

???

```
def add_stuff(a):  
    x = a + 5  
    return x  
  
b = 4  
print("Statement 1: The value of b is", b)  
y = add_stuff(7)  
print("Statement 2: The value of y is", y)  
print("Statement 3: The value of b is", b)
```

Note: Tools like pythontutor.com can be helpful in visualizing the concept of scope

Scope

- The area where a variable is recognized (or has meaning)
- **Global Variables** - declared outside of a function.
 - Have scope from where they are first created until the end of the program (or module/file)
- **Local Variables** - declared inside of a function.
 - Have scope from where they are first created until the end of the function definition
 - This includes parameters
- In this example, how would you describe the scope of the variable x?
 - a) x is a local variable
 - b) It is local to function add_stuff
- What about variable b?
 - a) b is a global variable

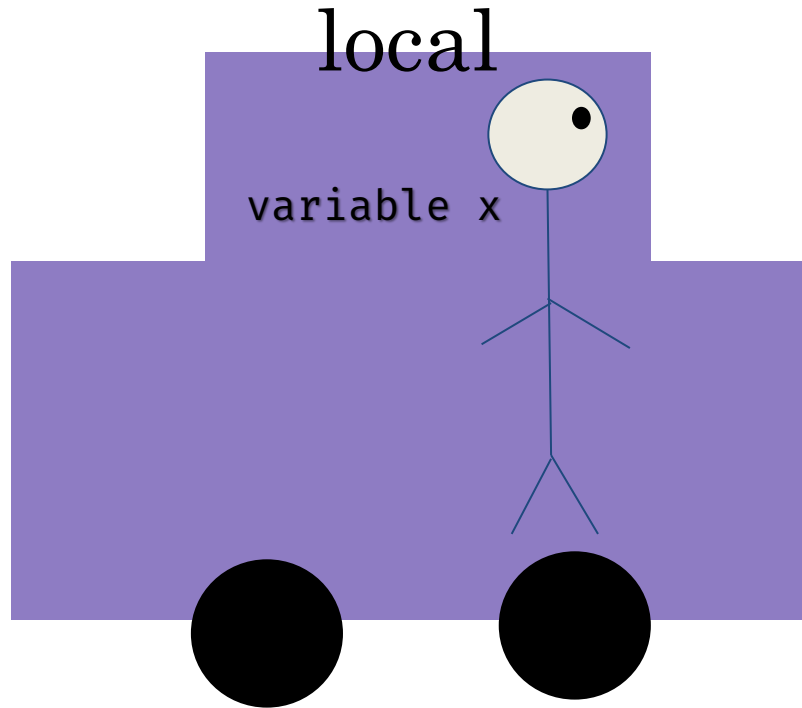
```
def add_stuff(a):  
    x = a + 5  
    return x  
  
b = 4  
print("Statement 1: The value of b is", b)  
y = add_stuff(7)  
print("Statement 2: The value of y is", y)  
print("Statement 3: The value of b is", b)
```

Note: Tools like pythontutor.com can be helpful in visualizing the concept of scope

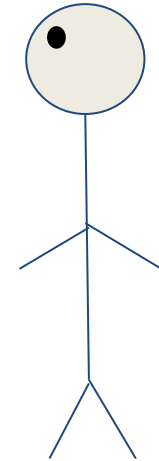
Variables with the same name but different scopes: Which variable will be used?

- If the **variable name** is on the **left** of an assignment operator (=) - **if** it *does not have a value yet*:
 - Python treats it as a **local variable** unless it has been *explicitly* declared with the **global** operator.
 - If the variable has been explicitly declared with the **global** operator, the global variable will be used.
- If the variable is assumed to *have a value already*:
 - If a **local variable** by that name exists, the **local variable** will be used
 - If **no local variable** by that name exists, the **global variable** will be used
- The keyword **global** can be used within a function to make scope explicit

Kind of like a car with tinted windows



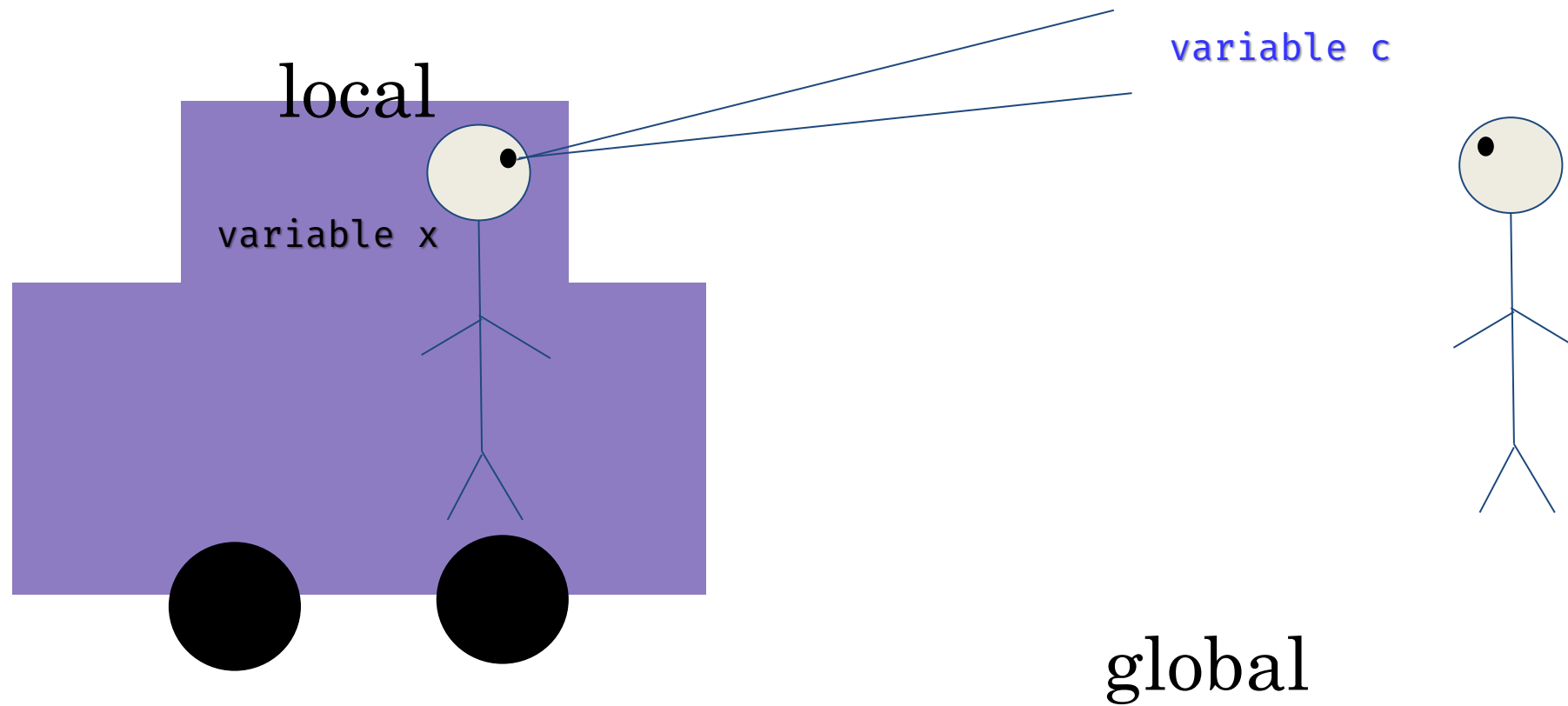
variable c



global

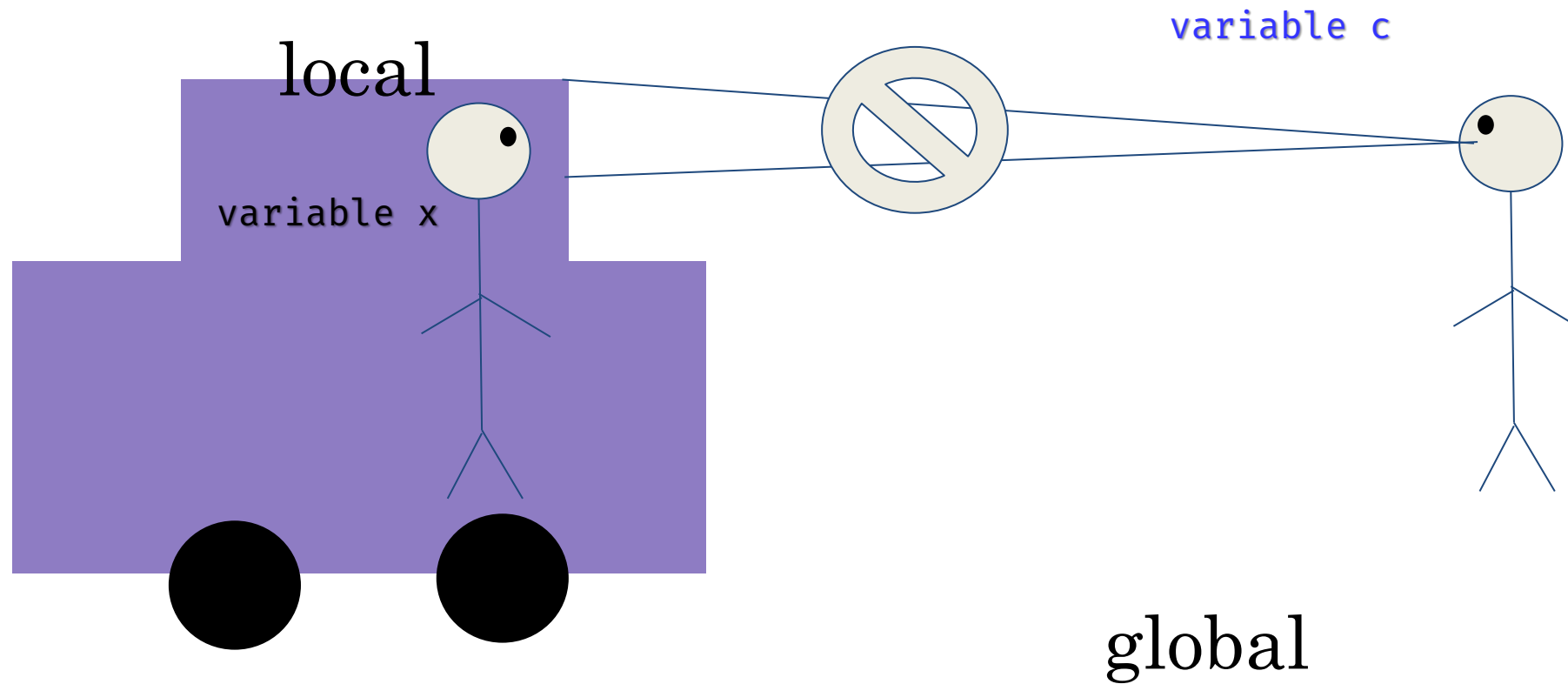
Kind of like a car with tinted windows

You can see variables (like c) outside the car from inside the car



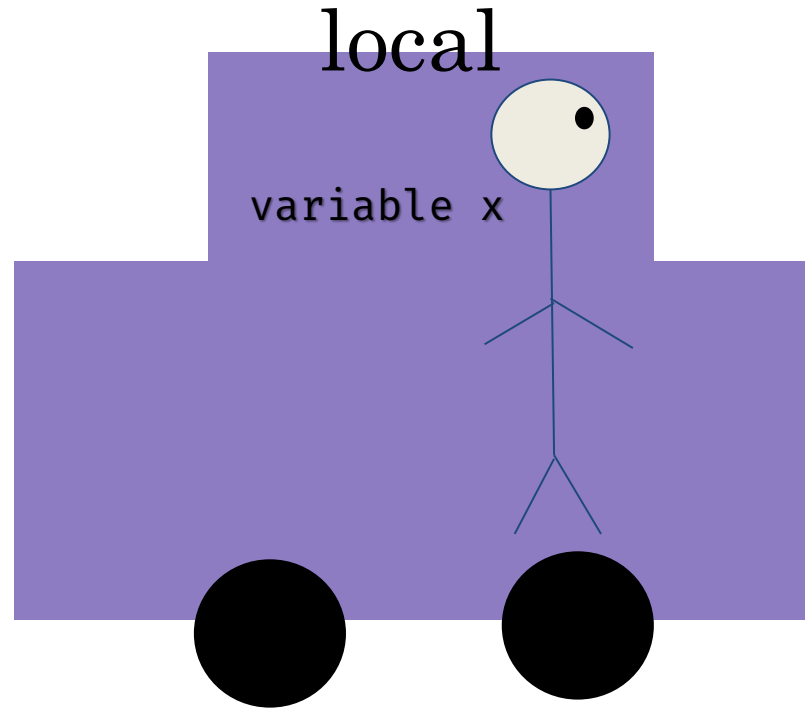
Kind of like a car with tinted windows

You cannot see variables inside the car (like x) from outside the car (tinted windows)

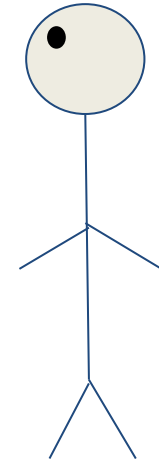


Kind of like a car with tinted windows

To modify variables outside the car, you must ask your friend to help you (calling global)




variable c



global

Which Variable is in Scope?


Q1: Variable **x** - Are these two different variables or are they the same variable?



```
def add_stuff(a):  
    result = x + 5  
    return result
```

```
x = 4  
y = x + 3
```


Q2: Variable **c** - Are these two different variables or are they the same variable?



```
def add_stuff(p):  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```

Q3: The keyword **global** - Does this change anything?



```
def add_stuff(p):  
    global c  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```

Which Variable is in Scope?

Q1: Variable **x** - Are these two different variables or are they the same variable?

These are the same variable

```
def add_stuff(a):  
    result = x + 5  
    return result
```

```
x = 4  
y = x + 3
```

Q2: Variable **c** - Are these two different variables or are they the same variable?

```
def add_stuff(p):  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```

Q3: The keyword **global** - Does this change anything?

```
def add_stuff(p):  
    global c  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```

Which Variable is in Scope?

Q1: Variable **x** - Are these two different variables or are they the same variable?

These are the same variable

```
def add_stuff(a):  
    result = x + 5  
    return result
```

```
x = 4  
y = x + 3
```

Q2: Variable **c** - Are these two different variables or are they the same variable?

These are two different variables

```
def add_stuff(p):  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```

Q3: The keyword **global** - Does this change anything?

```
def add_stuff(p):  
    global c  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```

Which Variable is in Scope?

Q1: Variable **x** - Are these two different variables or are they the same variable?

These are the same variable

```
def add_stuff(a):  
    result = x + 5  
    return result
```

```
x = 4  
y = x + 3
```

Q2: Variable **c** - Are these two different variables or are they the same variable?

These are two different variables

```
def add_stuff(p):  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```

Q3: The keyword **global** - Does this change anything?

Now these are the same variable

```
def add_stuff(p):  
    global c  
    c = p + 5  
    return c
```

```
c = 4  
c = c + 3
```



PYTHON DEMONSTRATION

Let's jump on PyCharm!

`more_function_basics.py`

Activity on Scope

- In **pairs** or groups **up to three** work on the following activity.
- **Variable Scope In-Class Activity**
- *Practice identifying variable scope.
Practice tracing code in a function.*

Remember to **check-in** with a TA
before leaving class today!

In-Class “lab” Activity!


```

# goal of this function is to double a number
# input: the number that I want doubled
# output (return value): the doubled number
def my_function(x): # this is assigning the value of the
    # passed-in argument to the variable x
    x_doubled = x + x
    return x_doubled

# remember to typecast input if you want something other than a string
my_num = input("Enter a number: ")
doubled_my_num = my_function(float(my_num))
print("When we double", my_num, "we get", doubled_my_num)

#
#
# in the computer's memory, this is what's going on when we create
# variables....
#
# variable name | variable type | scope | variable address
# -----
# my_num | string | global | 0xFA724B
# doubled_my_num | float | global | 0xA73D91
# x | float | my_function | 0x453AB1

# and example of a function that doesn't return anything
def silly_function(): # no parameters
    print("this is just a print line in the silly function")
    return
result = silly_function()
print("the silly function returned:", result)
# notice that the return is None

```

```

# since we ask the user to enter an integer often, we could create a
# function that would allow us to put the call to input, the type
# casting, and more (in the future, we'll think about input validation)
# all in one place
def get_int(the_prompt):
    # ... we could have more code here to make sure the number is valid
    in_num = int(input(the_prompt))
    return in_num

large = get_int("Enter a large number: ")
print(large * 2, "is larger than", large)

# you can always experiment with Python to learn the language better
# In this example, we are experimenting with types and operators
# try various operators on different pairings and orderings of
# a string and an int
#
str1 = "hello"
int1 = 9
result = str1 * int1 # swap the order, use different operators, etc.
print(type(result))
print(result)

# in the future, we'll discuss putting a function like this in a
# different # file than the one we are currently working on. Then we can
# call it
# anytime we want by using import

```

Review this code on your own.
Don't hesitate to ask the TAs or the
professor questions if you have any!

```

# The goal of this function is to double a number.
# input: the number that I want doubled
# output (return value): the doubled number

# Think: my_function(9) -> implicit assignment x = 9
def my_function(x): # function header: my_function is the function
name,
    # and x is the only parameter.
    # This is the function body (note code is indented), executed when
    # function is called. This code does NOT get executed until the
    # function is called/invoked.
    y = x+x
    return y # control of the program will return to wherever we called
    # this function from, and 'return' y to tell that part of
    # the code.
    # When a function returns a value, think:
    # "replace the function call with the returned value"

# When you are done with the function definition, just
# de-indent(un-indent?). You Can use backspace or shift+tab.

my_num = input("give me a number! ") # input() is a function too, and
    # "give me a number" is its argument here. The input function always
    # returns a string. For example, if the user only types in digits,
    # those digits are returned as a string.

my_num = int(my_num) # int() is a function that takes in one argument
    # and returns a value of type int

print(my_function(my_num)) # my_function() gets called with a number
    # passed in (an argument). When the function returns, the function
    # call becomes that returned value.

```

Review this code on your own.
 Don't hesitate to ask the TAs or the
 professor questions if you have any!

Review/Practice: Writing Functions

Practice writing a function. Do this activity on your own or find a friend in the class to work with. **There is nothing to submit, and you do not need to check-in with a TA** (this particular activity is *not* an in-class “lab” activity.)

You are welcome to discuss this function with TAs or your professor during office hours, or ask about it on Piazza)

Review: Writing Functions

- Write a **function** called `calculate_meal_cost` that has in three (3) parameters
 - 1 parameter for the original cost of the meal
 - 1 parameter for the amount of tip (pre-tax) (a percentage)
 - 1 parameter for the amount of tax (a percentage)
- What would the **function header** look like?

Review: Writing Functions

- Write a **function** called `calculate_meal_cost` that has in three (3) parameters
 - 1 parameter for the original cost of the meal
 - 1 parameter for the amount of tip (pre-tax) (a percentage)
 - 1 parameter for the amount of tax (a percentage)
- What would the **function header** look like?

```
def calculate_meal_cost(orig_cost, tip_pct, tax_pct):
```

Review: Writing Functions

- Write a **function** called `calculate_meal_cost` that has in three (3) parameters
 - 1 parameter for the original cost of the meal
 - 1 parameter for the amount of tip (pre-tax) (a percentage)
 - 1 parameter for the amount of tax (a percentage)
- What would the **function body** look like?
 - The tax amount will be the tax percent multiplied by the original cost
 - The tip amount will be the tip percent multiplied by the original cost
 - Then we should add all three of them together (original cost + tax amount + tip amount)

Header

```
def calculate_meal_cost(orig_cost, tip_pct, tax_pct):
```

Function call

```
calculate_meal_cost(90, 15, 10)
```

Output: 112.5

Review: Writing Functions (Possible Solution)

- The function body may look something like the following:

```
def calculate_meal_cost(orig_cost, tip_pct, tax_pct):  
    tip_amount = tip_pct / 100 * orig_cost  
    tax_amount = tax_pct / 100 * orig_cost  
    total_cost = orig_cost + tip_amount + tax_amount  
    return total_cost
```

- (*Don't forget to*: add comments including a multi-line function docstring)

Reminder: CS Laptop Loaner Program

- This course requires students to have a **laptop**
- I realize that not everybody might have one (nor necessarily need one for their desired major / path...)
- If you do not have a laptop for any reason... *not to worry!*
- The CS department's Systems staff has a notebook / laptop loaner program and will be able to loan you a notebook / laptop computer for the duration of the semester if you don't have one or if you cannot afford one.
 - Also available if your laptop is broken and under repair, we can arrange for you to receive a loaner laptop for a week or two until your own laptop is fixed

Interested? Link: https://www.cs.virginia.edu/wiki/doku.php?id=cs_laptop_loaner

I am happy to be your sponsor. Please let me know.