CS65: Introduction to Computer Science

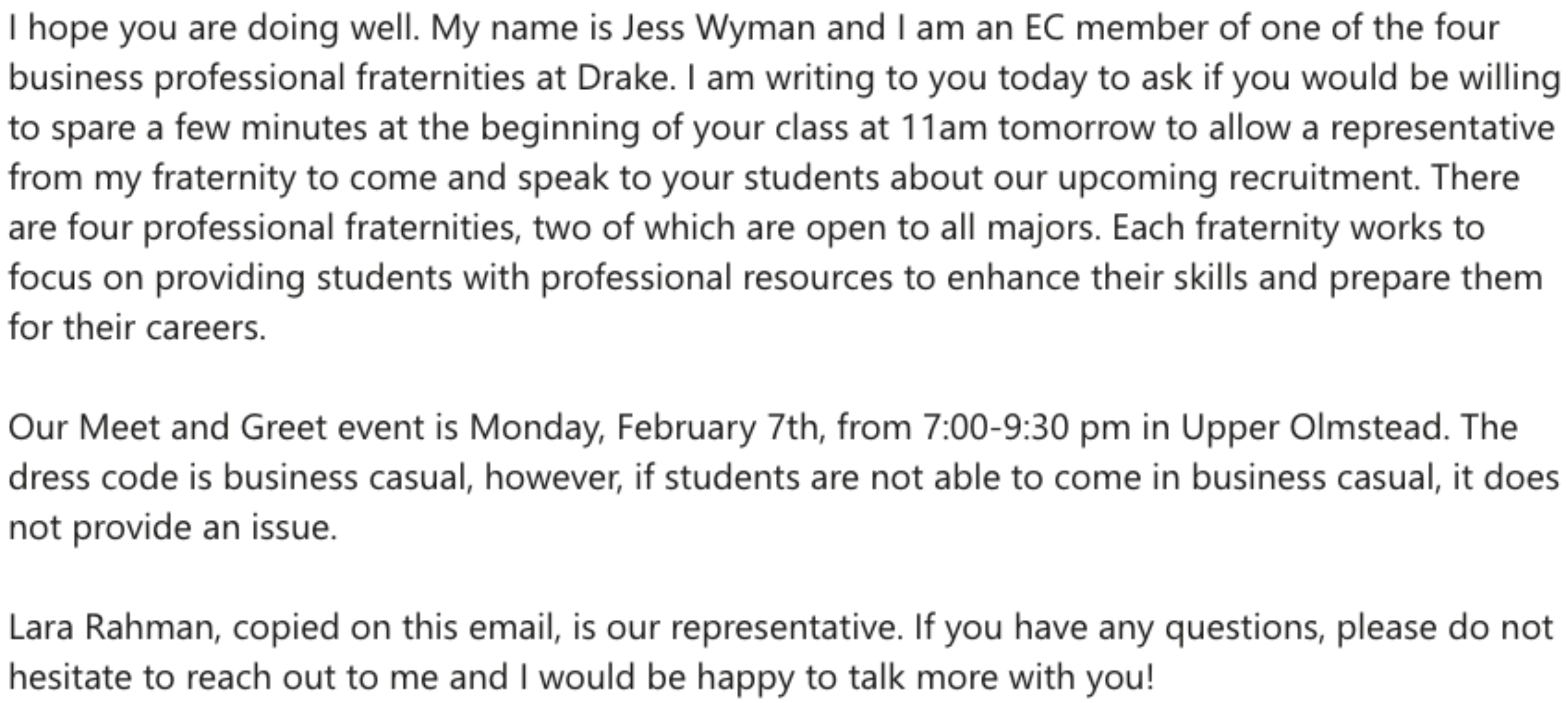
Built-in functions
Control flow during function call
Scope of a variable

February 03, 2022

Md Alimoor Reza
Assistant Professor of Computer Science

Drake
UNIVERSITY

Announcement



I hope you are doing well. My name is Jess Wyman and I am an EC member of one of the four business professional fraternities at Drake. I am writing to you today to ask if you would be willing to spare a few minutes at the beginning of your class at 11am tomorrow to allow a representative from my fraternity to come and speak to your students about our upcoming recruitment. There are four professional fraternities, two of which are open to all majors. Each fraternity works to focus on providing students with professional resources to enhance their skills and prepare them for their careers.

Our Meet and Greet event is Monday, February 7th, from 7:00-9:30 pm in Upper Olmstead. The dress code is business casual, however, if students are not able to come in business casual, it does not provide an issue.

Lara Rahman, copied on this email, is our representative. If you have any questions, please do not hesitate to reach out to me and I would be happy to talk more with you!

Recap

- Functions — a new concept
    - User defined functions vs built-in functions

- User defined functions
    - <u>defining</u> function: what statements it will execute
    - <u>calling</u> function: invoke/execute the defined body

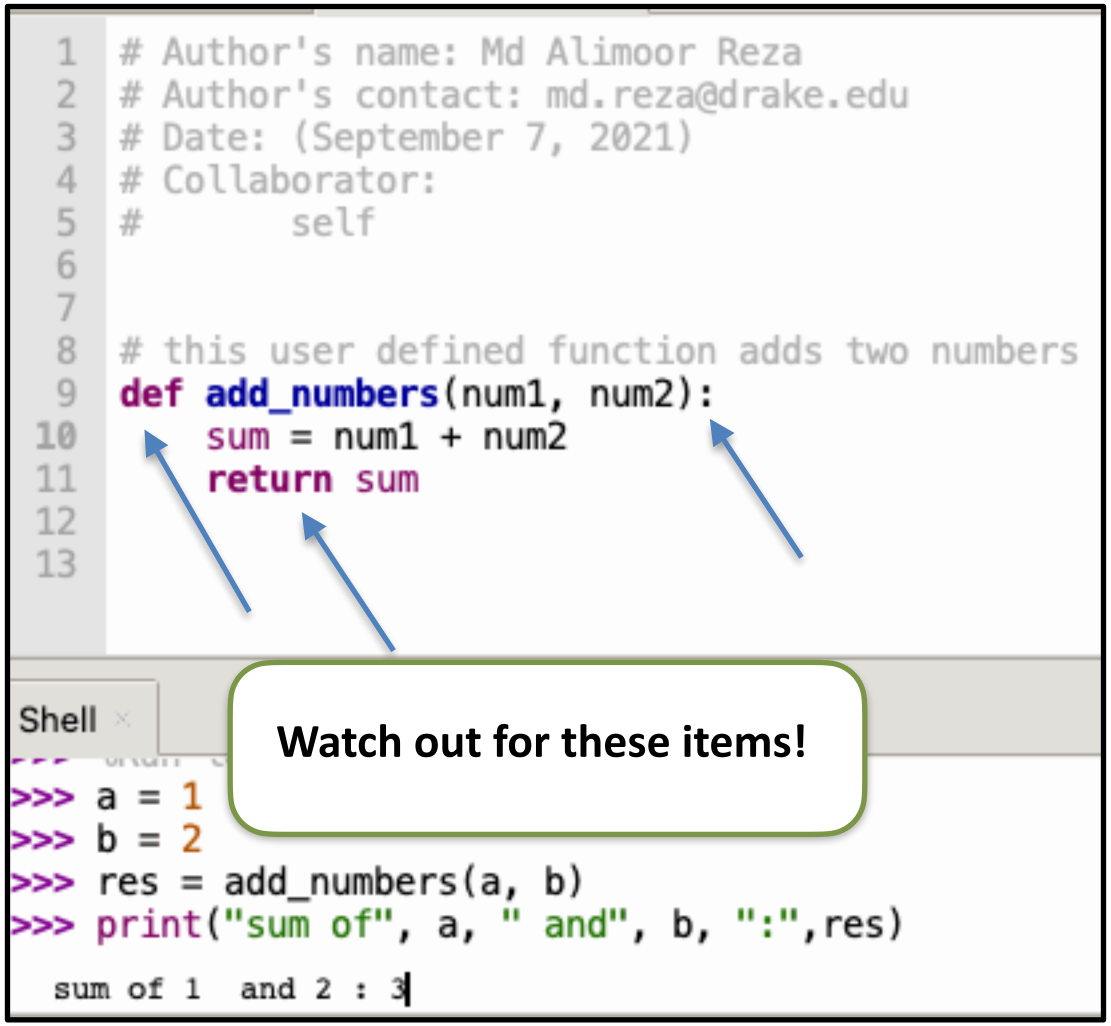
Recap: Define a function with no parameters

```
def name_of_the_function() :

        statement 1
        statement 2
        …
        statement 100
        return expression
```

This line is called function header

- **name_of_the_function**: a meaningful name denoting the task with a preceding **def** keyword
- **statements**: a sequence of python instructions to be executed followed by an optional **return** keyword with expression(s)
    - without a **return** statement function implicitly returns **None**
- Notice: indention (eg, tab) is required to define a **function** and also notice at the end of the condition expression there is a **colon**

Recap: user defined function example



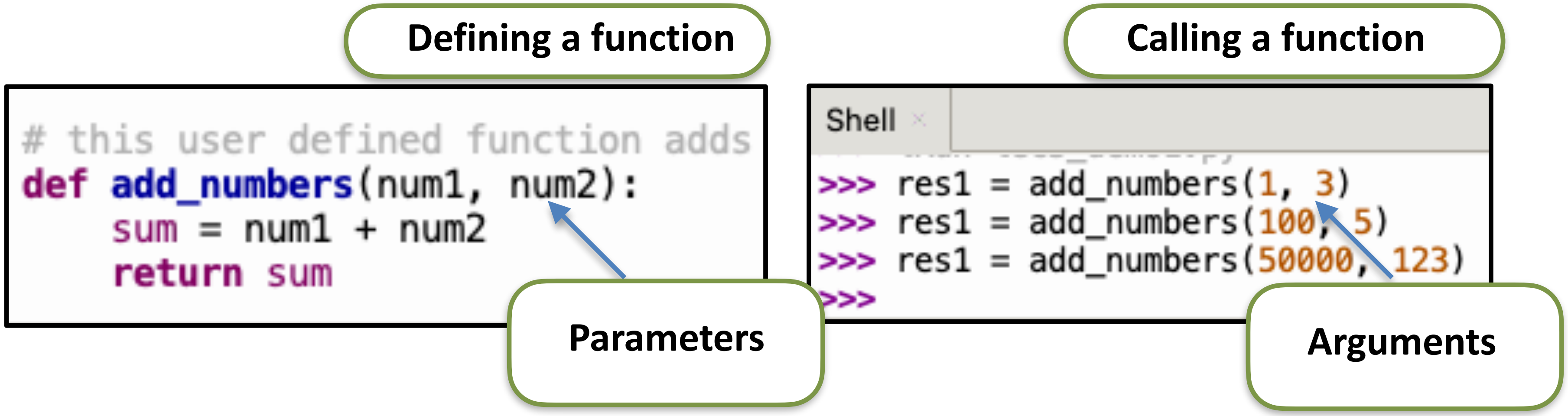
```
# Author's name: Md Alimoor Reza
# Author's contact: md.reza@drake.edu
# Date: (September 7, 2021)
# Collaborator:
#        self


# this user defined function adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum
```

Watch out for these items!

```
>>> a = 1
>>> b = 2
>>> res = add_numbers(a, b)
>>> print("sum of", a, " and", b, ":",res)

 sum of 1  and 2 : 3
```

Recap: calling a function

- **name_of_the_function**(*argument$_1$*, *argument$_2$*, …, *argument$_4$*)



```
# this user defined function adds
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum
```

```
>>> res1 = add_numbers(1, 3)
>>> res1 = add_numbers(100, 5)
>>> res1 = add_numbers(50000, 123)
>>>
```

- Function **calling name** should match function **definition name**
- Use *values*, *expression*, or *variables* to the **parameters** of the function
  - **arguments** should match **parameters**: one-to-one mapping
- When you call the function the execution gets transferred to the statements inside the function definition

Topics

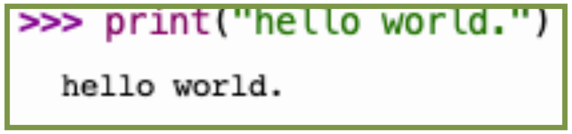
- Built-in functions in Python
    - eg, ***input***() receiving input from user
    - No need to define, just call

- Control flow during function call
    - Debugging features of Thonny
    - Step-by-step execution of your program

- Scope of a variable
    - Global scope vs local scope

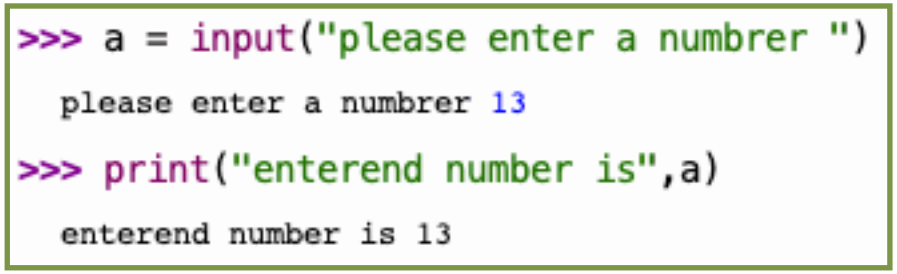
Built-in functions in Python

- You **do not** need to **define** the function; just call it
- We have already used 3 built-in functions:
  - ***print***()



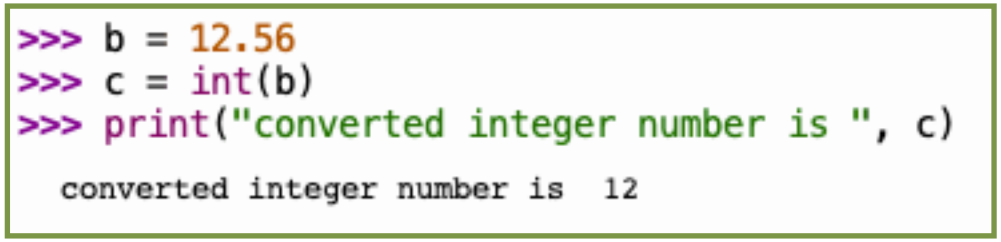
```
>>> print("hello world.")
  hello world.
```

  - ***input***()



```
>>> a = input("please enter a numbrer ")
  please enter a numbrer 13
>>> print("enterend number is",a)
  enterend number is 13
```

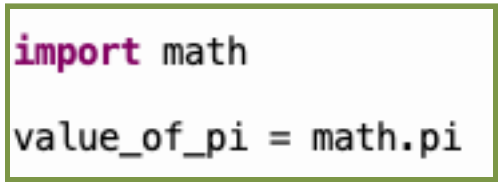
  - ***int***()



```
>>> b = 12.56
>>> c = int(b)
>>> print("converted integer number is ", c)
  converted integer number is  12
```

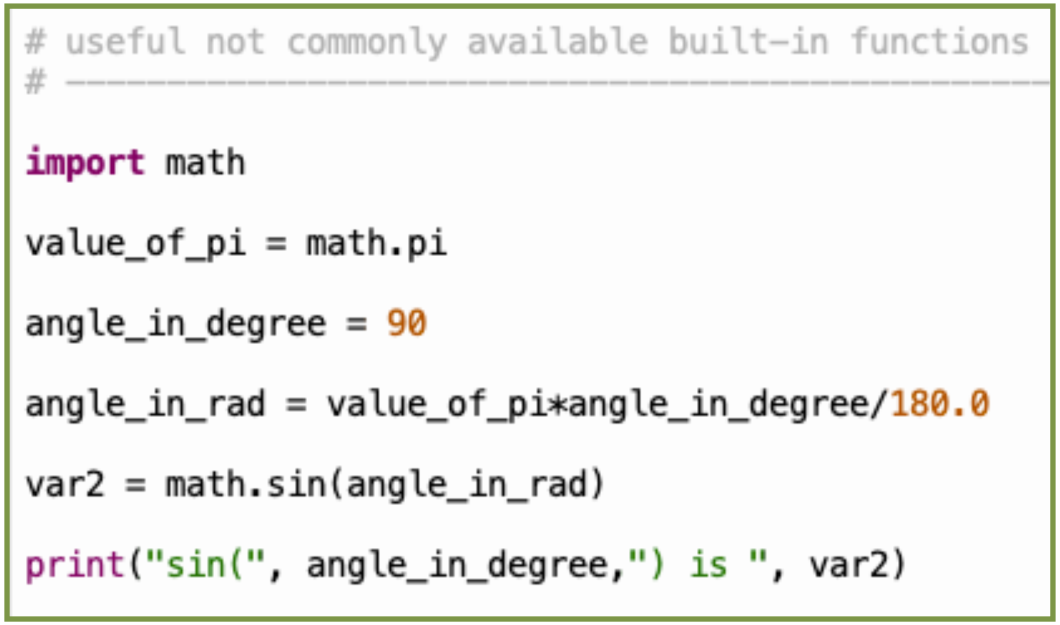
Built-in functions in Python

- If you want to use not so commonly available built-in functions, those built-in functions need to be imported using import keyword from a library
    - library also called a <u>module</u>
- Import the module before using it usually at the top of your python file
- Call function using *module_name* . *function_name*

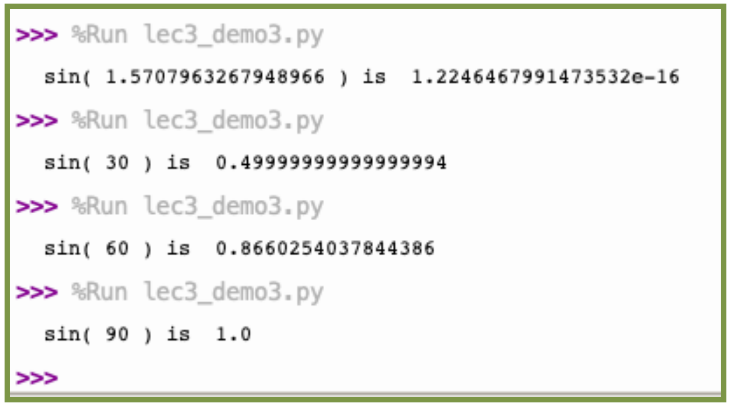


```python
import math

value_of_pi = math.pi
```

Built-in functions in Python



```
# useful not commonly available built-in functions
# ----------------------------------------------------

import math

value_of_pi = math.pi

angle_in_degree = 90

angle_in_rad = value_of_pi*angle_in_degree/180.0

var2 = math.sin(angle_in_rad)

print("sin(", angle_in_degree,") is ", var2)
```



```
>>> %Run lec3_demo3.py
  sin( 1.5707963267948966 ) is  1.2246467991473532e-16
>>> %Run lec3_demo3.py
  sin( 30 ) is  0.49999999999999994
>>> %Run lec3_demo3.py
  sin( 60 ) is  0.8660254037844386
>>> %Run lec3_demo3.py
  sin( 90 ) is  1.0
>>>
```

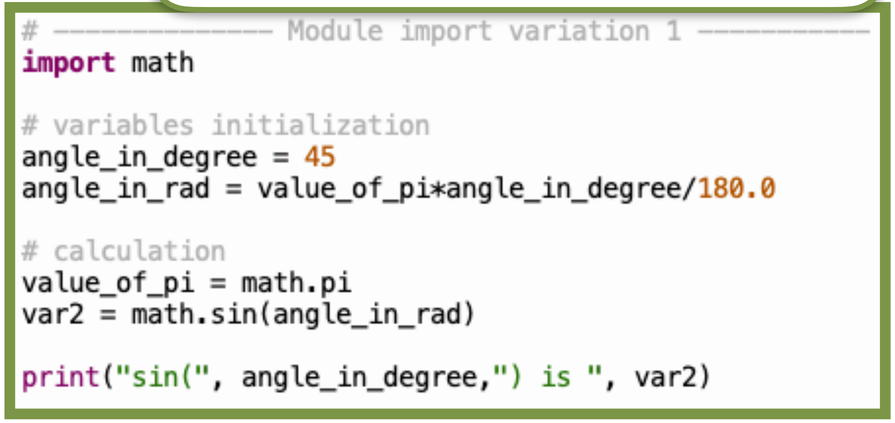
Module

- Formally, a module is a component containing Python functions, variables or class
- Each python file (with *.py) is a module
- They need to be imported from a module using import
    - Several ways of importing module components

https://docs.python.org/3/tutorial/modules.html

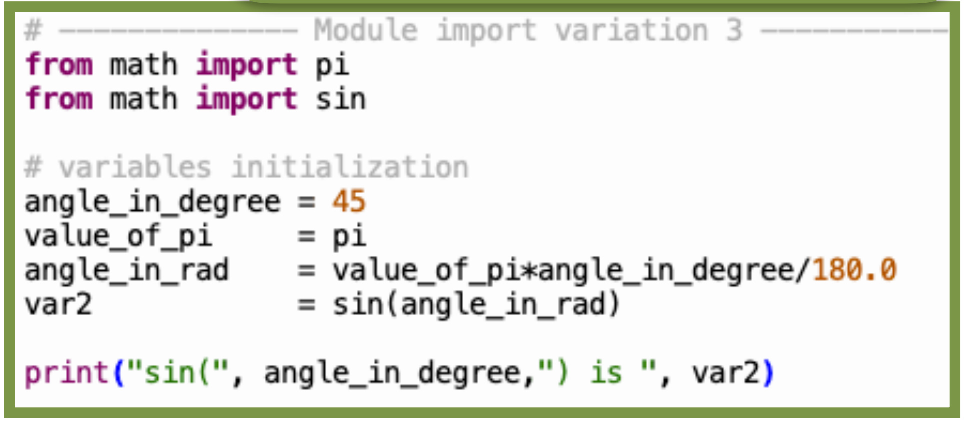
Module import variations

Explicitly need to use ***math.pi*** or ***math.sin***

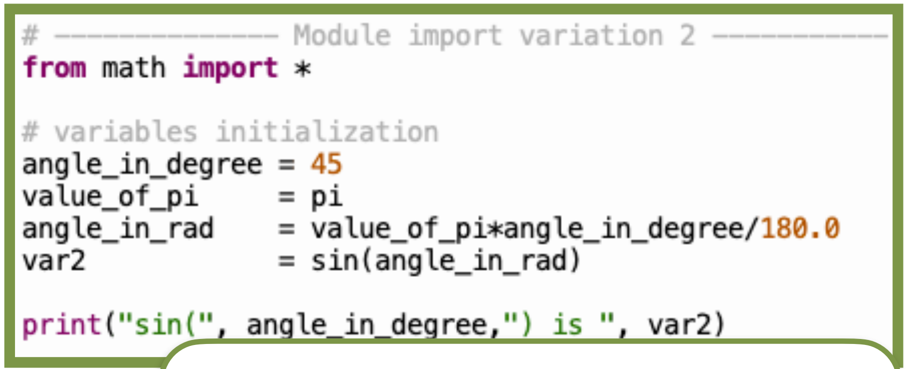


```
# -------------- Module import variation 1 -----------
import math

# variables initialization
angle_in_degree = 45
angle_in_rad = value_of_pi*angle_in_degree/180.0

# calculation
value_of_pi = math.pi
var2 = math.sin(angle_in_rad)

print("sin(", angle_in_degree,") is ", var2)
```

Directly access ***pi*** and ***sin*** *but nothing else*

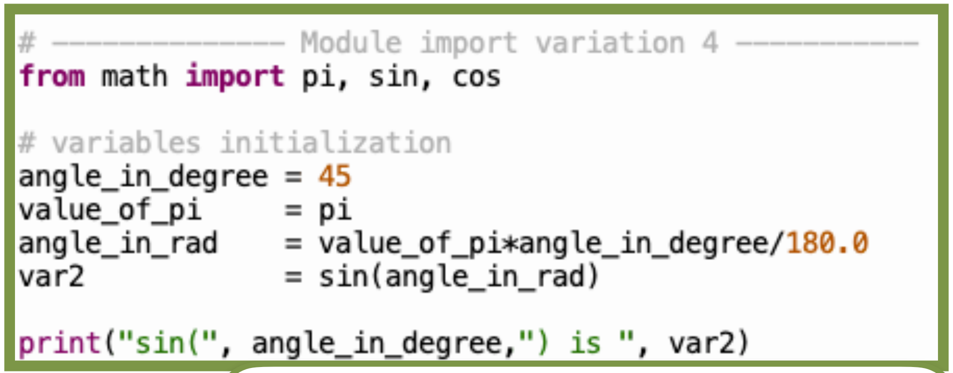


```
# -------------- Module import variation 3 -----------
from math import pi
from math import sin

# variables initialization
angle_in_degree = 45
value_of_pi     = pi
angle_in_rad    = value_of_pi*angle_in_degree/180.0
var2            = sin(angle_in_rad)

print("sin(", angle_in_degree,") is ", var2)
```



```
# -------------- Module import variation 2 -----------
from math import *

# variables initialization
angle_in_degree = 45
value_of_pi     = pi
angle_in_rad    = value_of_pi*angle_in_degree/180.0
var2            = sin(angle_in_rad)

print("sin(", angle_in_degree,") is ", var2)
```

Directly access ***pi*** or ***sin***



```
# -------------- Module import variation 4 -----------
from math import pi, sin, cos

# variables initialization
angle_in_degree = 45
value_of_pi     = pi
angle_in_rad    = value_of_pi*angle_in_degree/180.0
var2            = sin(angle_in_rad)

print("sin(", angle_in_degree,") is ", var2)
```

Directly access ***pi*** ***sin** and* ***cos*** *(in a single import line) but nothing else*

https://docs.python.org/3/tutorial/modules.html

Topics

- Built-in functions in Python
    - eg, ***input***() receiving input from user
    - No need to define, just call

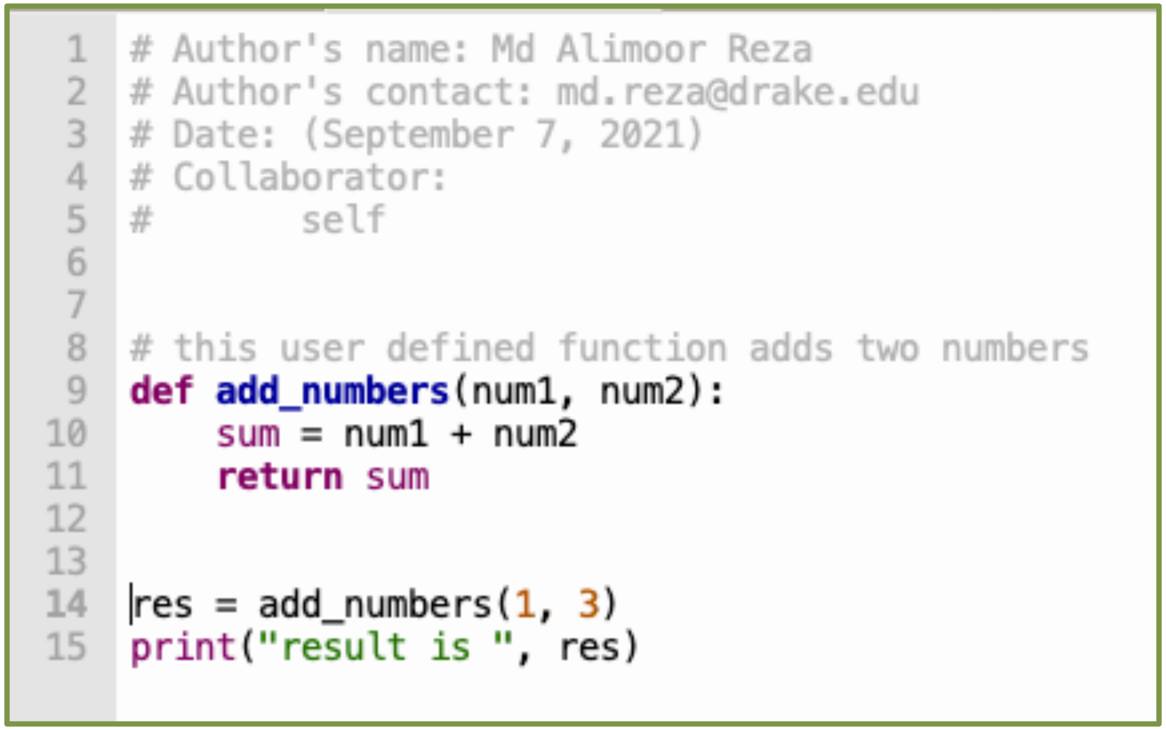
- Control flow during function call
    - Debugging features of Thonny
    - Step-by-step execution of your program

- Scope of a variable
    - Global scope vs local scope

Control flow during function call

- Should **define** a function before you can **call** the function
- A function can be called from anywhere, even from another function
- Sequence of steps during a function call:
  - <u>Step 1:</u> call a function
  - <u>Step 2:</u> go to the start of that function (inside the function definition)
    - caller is paused
  - <u>Step 3:</u> execute the instructions inside the function
  - <u>Step 4:</u> control goes back to where it's called from

Demo: Control flow during function call



```python
# Author's name: Md Alimoor Reza
# Author's contact: md.reza@drake.edu
# Date: (September 7, 2021)
# Collaborator:
#       self


# this user defined function adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum


res = add_numbers(1, 3)
print("result is ", res)
```

Sequence of steps during a function call:

- Step 1: call a function
- Step 2: transfer to the start of the function (caller is paused)
- Step 3: Execute the instruction inside the function
- Step 4: jump back to where the function was called from

Demo: debug feature of Thonny

Topics

- Built-in functions in Python
    - eg, ***input***() receiving input from user
    - No need to define, just call
- Control flow during function call
    - Debugging features of Thonny
    - Step-by-step execution of your program
- Scope of a variable
    - Global scope vs local scope

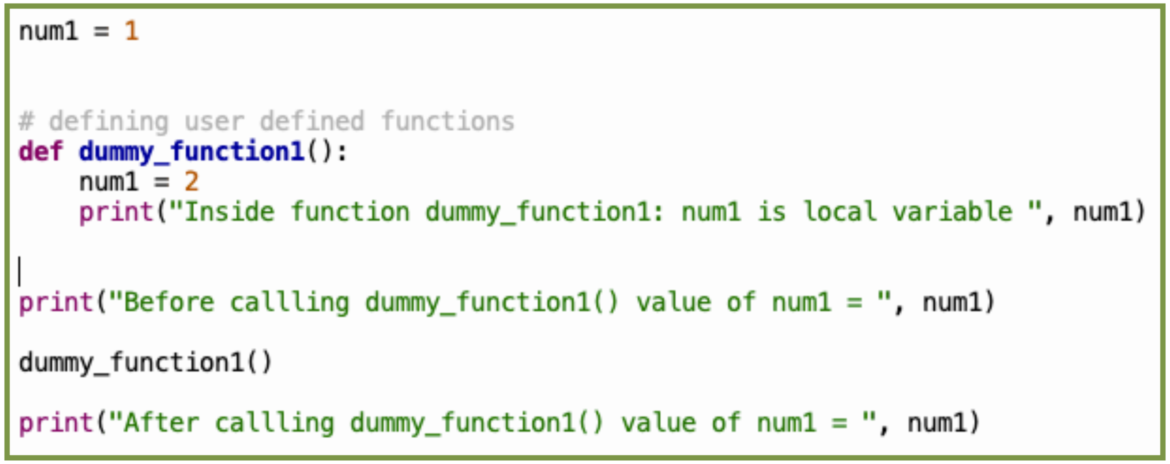
Local and global variables

- Local variables:
  - Variables declared 1) inside function 2) function parameters
  - Only visible to the defined function

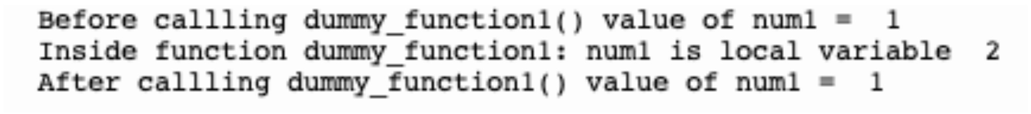
- Global variables:
  - Variables that are defined outside of user defined functions
  - Can be accessed by any function after creation
  - Global variable can be replaced/hidden by local variable if declared with the same name

Scope: local and global variables

- Global variables:
    - Variables that are defined outside of user defined functions
    - Can be accessed by any function after creation
    - Global variable can be replaced/hidden by local variable if declared with the same name



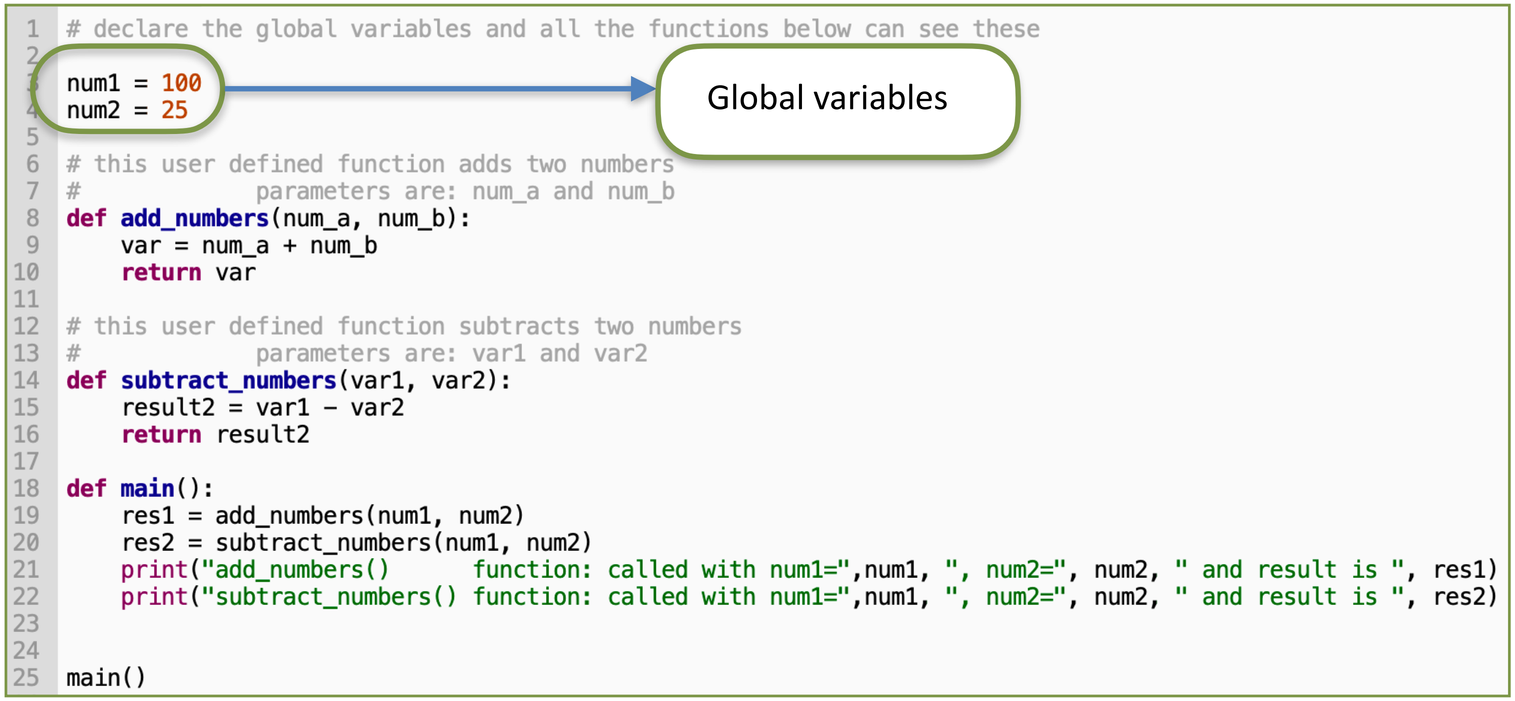
```
num1 = 1


# defining user defined functions
def dummy_function1():
    num1 = 2
    print("Inside function dummy_function1: num1 is local variable ", num1)


print("Before callling dummy_function1() value of num1 = ", num1)

dummy_function1()

print("After callling dummy_function1() value of num1 = ", num1)
```



```
Before callling dummy_function1() value of num1 =  1
Inside function dummy_function1: num1 is local variable  2
After callling dummy_function1() value of num1 =  1
```

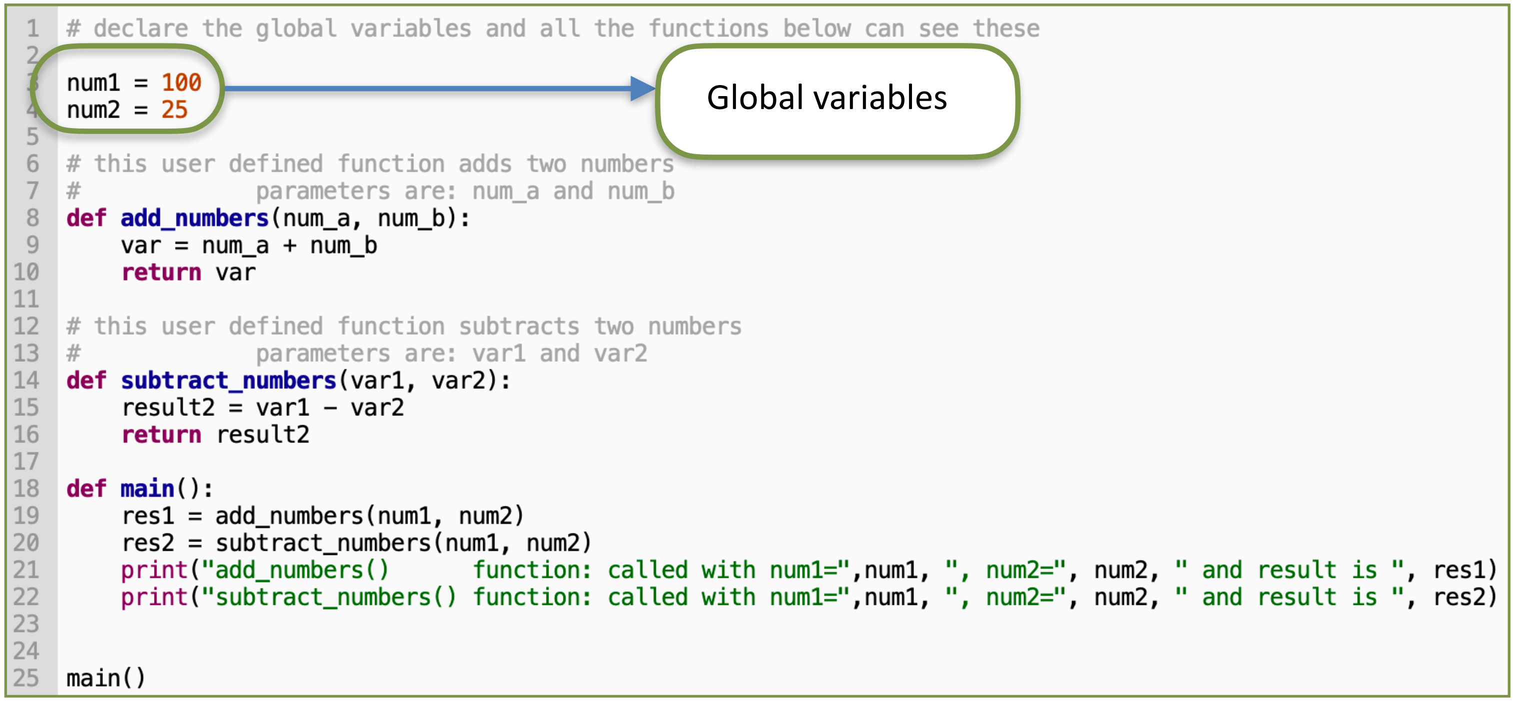
Scope: local and global variables

- <u>Global variables:</u>
  - Variables that are defined outside of user defined functions
  - Can be accessed by any function
  - Here values of global variables are copied to the parameters during function call



```python
# declare the global variables and all the functions below can see these

num1 = 100
num2 = 25

# this user defined function adds two numbers
#              parameters are: num_a and num_b
def add_numbers(num_a, num_b):
    var = num_a + num_b
    return var

# this user defined function subtracts two numbers
#              parameters are: var1 and var2
def subtract_numbers(var1, var2):
    result2 = var1 - var2
    return result2

def main():
    res1 = add_numbers(num1, num2)
    res2 = subtract_numbers(num1, num2)
    print("add_numbers()      function: called with num1=",num1, ", num2=", num2, " and result is ", res1)
    print("subtract_numbers() function: called with num1=",num1, ", num2=", num2, " and result is ", res2)


main()
```

Demo



```
# declare the global variables and all the functions below can see these

num1 = 100
num2 = 25

# this user defined function adds two numbers
#              parameters are: num_a and num_b
def add_numbers(num_a, num_b):
    var = num_a + num_b
    return var

# this user defined function subtracts two numbers
#              parameters are: var1 and var2
def subtract_numbers(var1, var2):
    result2 = var1 - var2
    return result2

def main():
    res1 = add_numbers(num1, num2)
    res2 = subtract_numbers(num1, num2)
    print("add_numbers()      function: called with num1=",num1, ", num2=", num2, " and result is ", res1)
    print("subtract_numbers() function: called with num1=",num1, ", num2=", num2, " and result is ", res2)


main()
```

Scope: local and global variables

- Scope resolution: Mechanism of searching for a name, e.g., variable or function

  - **Step 1:** search the referenced name in the local scope. If not found, then go to step 2
  - **Step 2:** search the referenced name in the global scope. If not found, then go to step 3
  - **Step 3:** If searched name is not found in either step 1 or step 2, then search in the built-in scope
  - **Step 4:** If not found in the above steps, then interpreter generates an Error message

Demo

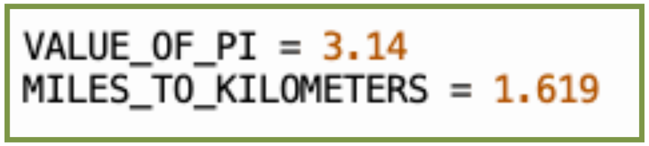
Global variables

- Global variables are defined outside of user defined functions or they can be introduced by the **global** statement

- As you have noticed by now, they can be source of confusion
  - Name clashing
  - Order of their definitions matter

- Use of global variables is not recommended, better to avoid or to minimize (at least) its usage

- If you need to use eg, some constants, then declare them using capital letters



```
VALUE_OF_PI = 3.14
MILES_TO_KILOMETERS = 1.619
```

Summary

- **Takeaway from this lecture:**
  - Calling Python's built-in functions (helper algorithms)
    - Don't need to reinvent the wheel, just use it!
  - Flow of functions can be observed in debug mode of Thonny
  - Global scope vs local scope of variables
    - Minimize the use of global variables

- **To do:**
  - Lab 2 released : due next Thursday 02/10/22
  - Finish reading— Chapter 3

- **Announcements:**
  - There will be a paper-based quiz next Tuesday 02/08/22
  - Open notes/slides but can't use Thonny