Module 1

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 1 Study Guide and Deliverables

Theme:	Introduction to Computing with Python
Readings:	• Chapter 1 (pp. 37-53), Chapter 9 (pp. 456-463), and Appendix A • Module Lecture Notes
Topics:	Introduction to Computing, Program Structure, Running Python, Input/Output, Variable Scopes and Modules
Assignments	Assignment 1 due on Tuesday, March 23 at 6:00 PM ET
Assessments	Quiz 1: • Available Friday, March 19 at 6:00 AM ET • Due on Tuesday, March 23 at 6:00 PM ET
Live Classrooms:	• Tuesday, March 16, 8:00 - 9:30 PM ET • Thursday, March 18, 6:00 - 7:30 PM ET • Facilitator Session: Friday, March 19, at 8:00 PM ET

Learning Objectives

After successfully completing this module, the learner is expected to do the following:

- Describe a basic structure of a Python program.
- Perform casting.
- Compare shell and script mode for running Python programs.
- Use input function.
- Apply Python conventions and syntax.
- Use Python indentation.
- Distinguish global and local scope.
- Use modules in Python.

Processing math: 100%

ion to Computing

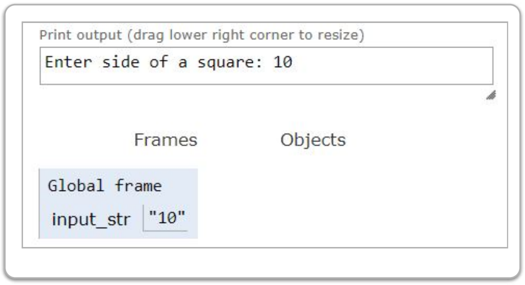
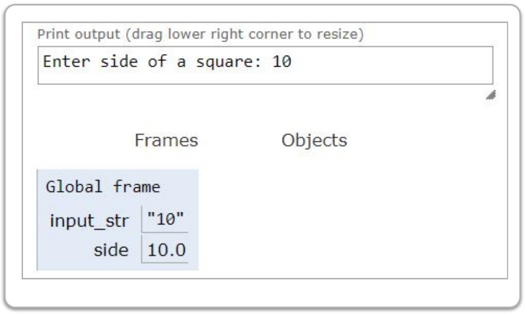
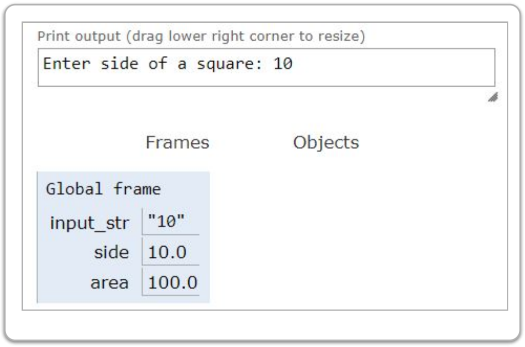
A Simple Program Example

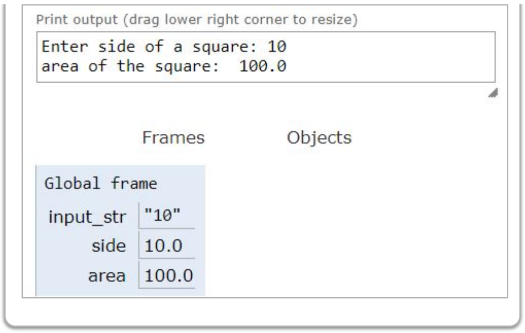
```
input_str = input('Enter side of a square: ')
side      = float(input_str)
area      = side * side
print('area of the square: ', area)
```

This simple program includes the following:

- get user input(s)
- perform computation(s)
- output result(s)

Let's go through the program line by line, to track its execution.

Code	Explanation	Program Execution Tracking
`input_str = input('Enter side of a square: ')`	input is a string	 Print output (drag lower right corner to resize) `Enter side of a square: 10` Frames Objects Global frame input_str "10"
`side = float(input_str)`	need to "cast" into numeric	 Print output (drag lower right corner to resize) `Enter side of a square: 10` Frames Objects Global frame input_str "10" side 10.0
`area = side * side`	computation of area	 Print output (drag lower right corner to resize) `Enter side of a square: 10` Frames Objects Global frame input_str "10" side 10.0 area 100.0
`print('area of the square: ', area)`	output of results	

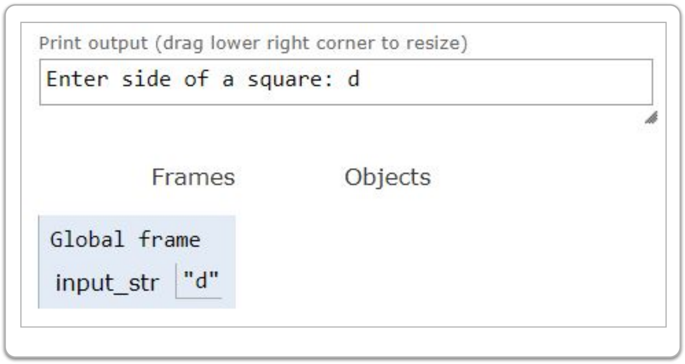
		
		Print output (drag lower right corner to resize)
		`Enter side of a square: 10`
		`area of the square:  100.0`
		Frames Objects
		Global frame
		input_str "10"
		side 10.0
		area 100.0

A Simple Program with Error

```
input_str = input('Enter side of a square: ')
side      = float(input_str)
area      = side * side
print ('area of the square: ', area)
```

This program calculates the area of the square based on the user input for the side length. If a user input a string such as "d":

- Cannot cast into numeric.
- Python will terminate.



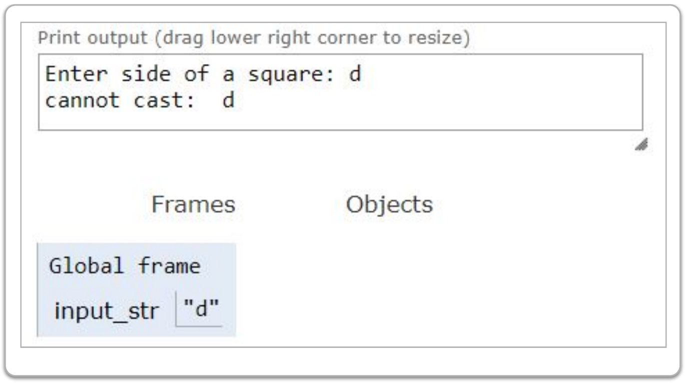
```
Enter side of a square: d
```

A Simple Program without Error

We can check if casting is possible by modifying the code:

```
input_str = input('Enter side of a square: ')
if input_str.isnumeric() is True:
     side = float(input_str)
     area = side * side
     print('area of the square: ', area)
else:
     print ('cannot cast: ', input_str)
```

Processing math: 100%



Print output (drag lower right corner to resize)

```
Enter side of a square: d
cannot cast:  d
```

Global frame

input_str	"d"

Computing: Test Yourself Exercises

The following are some exercise questions for you to practice. Please read each question, think carefully, click "Show hint" to review and relearn what you have learned, figure out your own answer or write your own program first, and then click "Show Answer" to compare yours to the suggested answer or the possible solution.

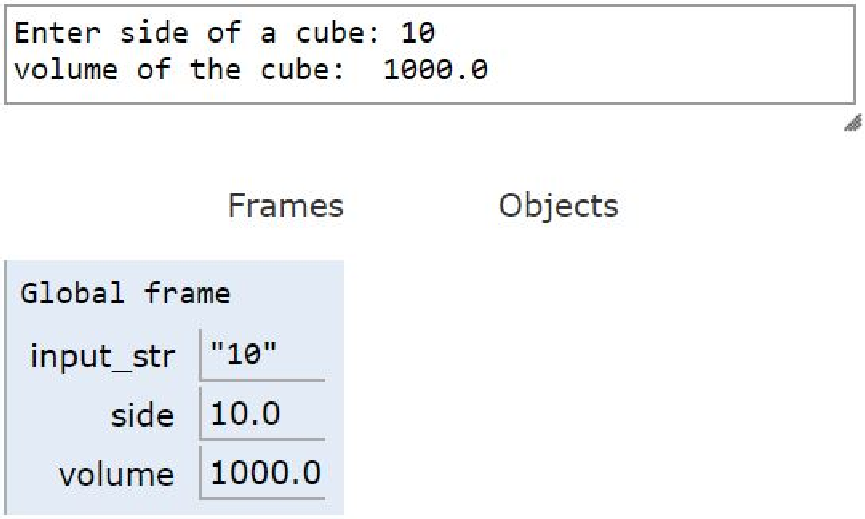
Test Yourself 1.01

Write a program that asks for the side of a cube and com- putes its volume.

▶ Show Hint

Suggested program:

```
input_str = input('Enter side of a cube: ')
side = float(input_str)
volume = side * side * side
print ('volume of the cube: ', volume)
```



```
Enter side of a cube: 10
volume of the cube:  1000.0
```

Global frame

input_str	"10"
side	10.0
volume	1000.0

Test Yourself 1.02

Write a program that asks for input and prints it three times.

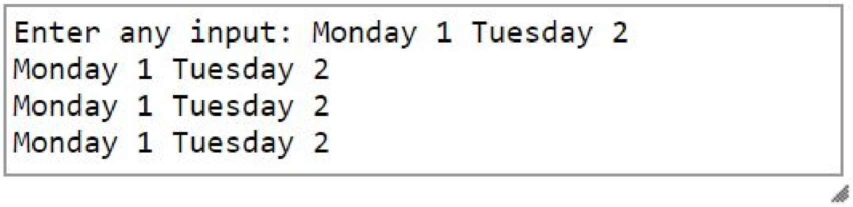
▶ Show Hint

Suggested program:

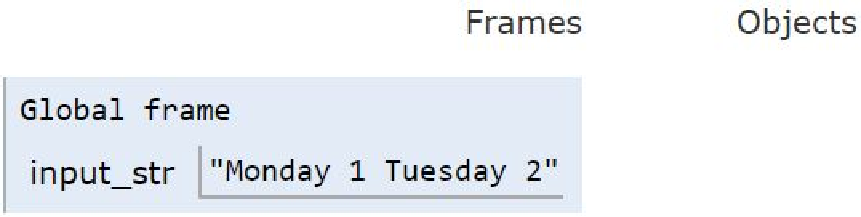
```
input_str = input('Enter any input: ')
t(input_str)
```

Processing math: 100%

```
print(input_str)
print(input_str)
```



```
Enter any input: Monday 1 Tuesday 2
Monday 1 Tuesday 2
Monday 1 Tuesday 2
Monday 1 Tuesday 2
```



Frames Objects

Global frame

input_str "Monday 1 Tuesday 2"

Test Yourself 1.03

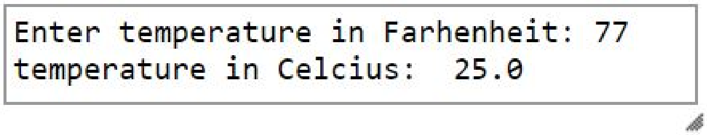
Write a program thet converts temperature in F Farhenheit to temperature in C in Celsius:

$$C = (F - 32). \frac{5}{9}$$

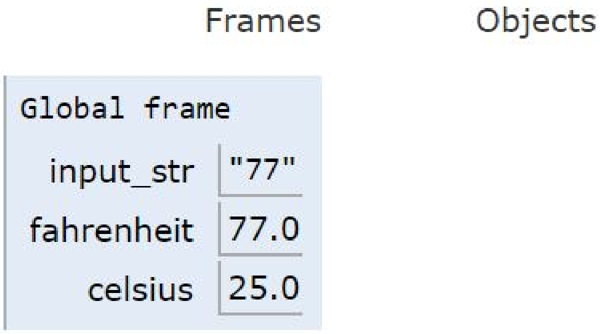
▶ Show Hint

Suggested program:

```
input_str = input('Enter temperature in Farhenheit fahrenheit = float(input_str)
celsius = (fahrenheit - 32) * (5/9)
print ('temperature in Celcius: ', celsius)
```



```
Enter temperature in Farhenheit: 77
temperature in Celcius:  25.0
```



Frames Objects

Global frame

input_str "77"

fahrenheit 77.0

celsius 25.0

Python Program Structure

- Python is an interpreted, high-level language.
- Python programs have extension *.py*
- There are two ways to run a program

Processing math: 100% mode

2. script mode

Shell Mode

Shell mode is also called interactive mode. The shell mode involves running the codes directly on the Python shell, which can be accessed from the command line shell. The shell mode gives immediate feedback for each statement—"interactive" use.

```
(base) C:\Users\epinsky>python
Python 3.6.3 |Anaconda, Inc.|
>>> input_str = input('Enter side of a square: ')
Enter side of a square: 10
>>> side = float(input_str)
>>> area = side * side
>>> print('area of the square: ', area)
area of the square: 100.0
>>>
```

Script Mode

In script mode, we need to write codes in a text file then save it with a *.py* extension that stands for "Python". For example, recall the simple program below we used earlier, it can be saved as "compute_area.py".

```
input_str = input('Enter side of a square: ')
if input_str.isnumeric () is True:
     side = float(input_str)
     area = side * side
     print('area of the square: ', area)
else:
     print('cannot cast ', input_str)
```

In the script mode, we run a *.py* file as script.

```
(base) C:\Users\epinsky>python compute_area.py
Enter side of a square: 10
area of the square: 100.0
(base) C:\Users\epinsky>
```

Program Structure Analogy

English: sentences

- building blocks (nouns, verbs, adjectives, adverbs)
- grammar (rules)

Python: statements

- data types (integers, strings, lists, sets, user-defined classes)
- syntax (rules)

Conventions and Syntax

- Program contains modules.

- Modules contain statements.
- Statements contain expressions.
- expressions create and process objects.
- Each statement ends with newline or continuation "\".
- Multiple statements per line separated by ";".
- Comments start with "#".

Python Types

Python types are the building blocks in a language, similar to noun, verb in English.

Python has two groups of types:

1. primitive types ("atoms")
2. collections ("molecules")

There are additional special types:

1. None type
2. range type

Variable Names

Rules of Python Variable Names:

- starts with letter or _
- **cannot** start with number
- case sensitive
- alphanumeric and _ only
- no reserved keywords

Examples:

Please review the following examples, examine why some are OK, some are illegal, and some are OK but not recommended.

```
assets      = 1000   # OK
_debts      = 500    # OK
for         = 100    # illegal ( reserved )
_for        = 150    # OK but not recommended
7 _lives    = 7      # illegal
two & three = 23     # illegal
```

Reserved Keywords

lowing reserved keywords as identifiers. Most of reserved keywords are lower case.

```
and       as        assert    break
class     continue  def       del
elif      else      except    False
finally   for       from      global
if        in        import    is
lambda    None      nonlocal  not
or        pass      raise     return
True      try       while     with
yield
```

Test Yourself Exercises

Test Yourself 1.04

Which of the following are not legal identifiers (Check all that are true.)

b = 4

This identifier is OK.

b 3 = 4.

Error: no spaces allowed.

b3 = 4

This identifier is OK.

b_3 = 4

This identifier is OK.

b-5 = 5

Error: no minus allowed

_b_3 = 4

This identifier is OK.

None = 4

Error: no reserved keyword.

b3$ = 4

Error: non alphanumeric $.

&b3 = 4

Error: non alphanumeric &.

▶ Show Hint

Variable Scopes

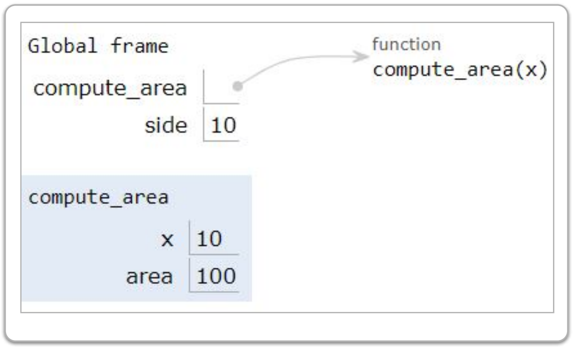
A variable is only available from inside the region it is created. This is called scope. There are local scope and global scope.

Local Scope

A variable created inside a function is available inside that function. The local variable can be accessed from a function within the function.

In the following example, variable "area" in compute area() function has local scope.

```
def compute_area(x):
    area = x * x
    return area

side = 10
area = compute_area(side)
```

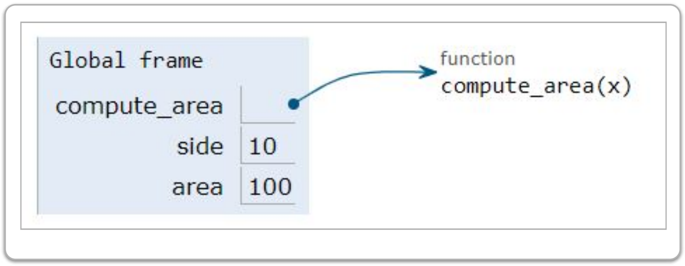


Global Scope

A variable created in the main program body, which is outside of a function, is global.

In the following example, variables "side" & "area" have global scope.

```
def compute_area(x):
    area = x * x
    return area

side = 10
area = compute_area(side)
```



Test Yourself Exercises

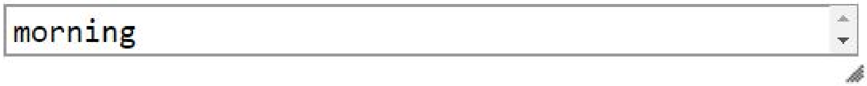
Test Yourself 1.05

What is the output of A?

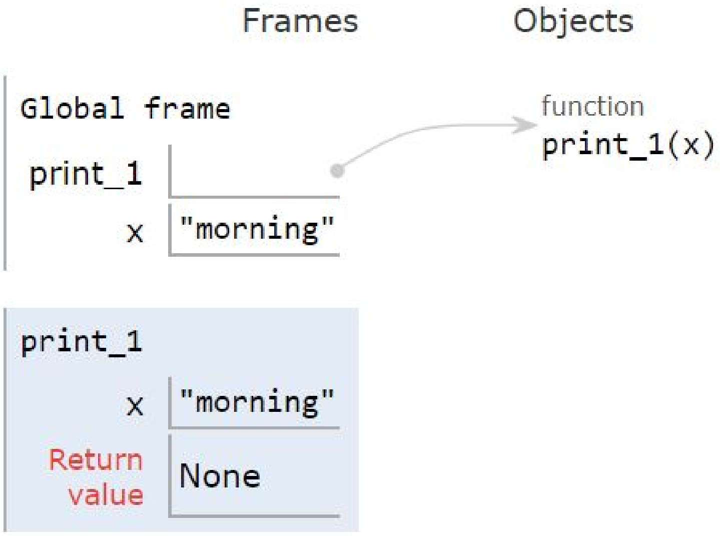
```
def print_1(x):
    print(x)

x = 'morning'
print_1(x)
```

▶ Show Hint

Suggested output:



```
morning
```



Test Yourself 1.06

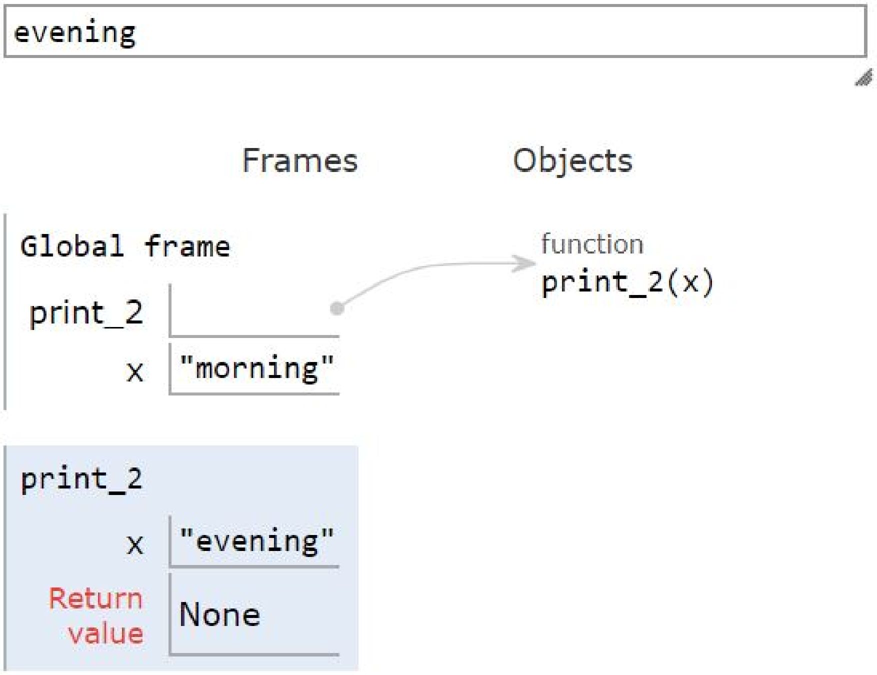
What is the output of B?

```
def print_2(x):
    x = 'evening'
    print(x)

x = 'morning'
print_2(x)
```

▶ Show Hint

Suggested output:

Processing math: 100%



```
evening
```

Test Yourself 1.07

Are the program outputs of A and B the same? Why?

▶ Show Hint

Suggested answer:

The outputs are different: in A we print the passed parameter "morning"; in B, we reset passed parameter "morning" to the new value "evening".

Indentation

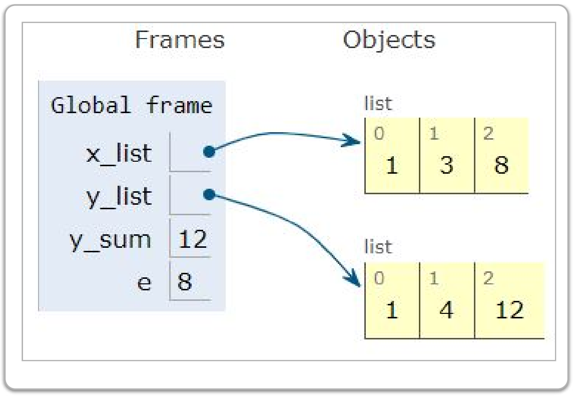
Indentation refers to the whitespaces at the beginning of a code line. Although the indentation in code in other programming languages is for readability only, the indentation in Python has very important meanings for real: to determine the grouping of statements.

Same lines of code with different indentation levels have different logical meanings and may output different results.

In the following program (Indentation Example 1), all sums are computed.

Indentation Example 1

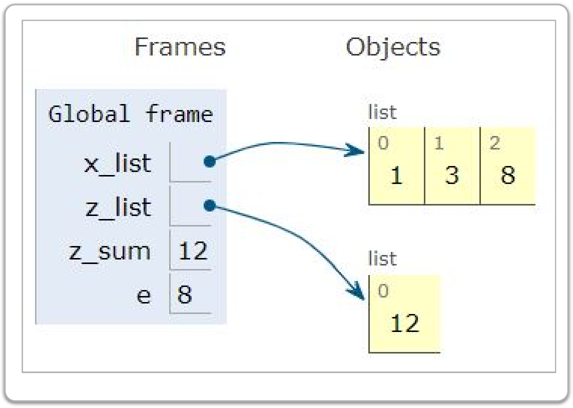
```
# list of cumulative sums
x_list = [1, 3, 8]

y_list = []
y_sum  = 0
for e in x_list :
    y_sum = y_sum + e
    y_list.append(y_sum)
```



In the following program (Indentation Example 2), only one sum is computed.

Indentation Example 2

```
# list of cumulative sums
x_list = [1, 3, 8]

z_list = []
z_sum  = 0
for e in x_list :
    z_sum = z_sum + e
z_list.append(z_sum)
```



Indentation Comparison

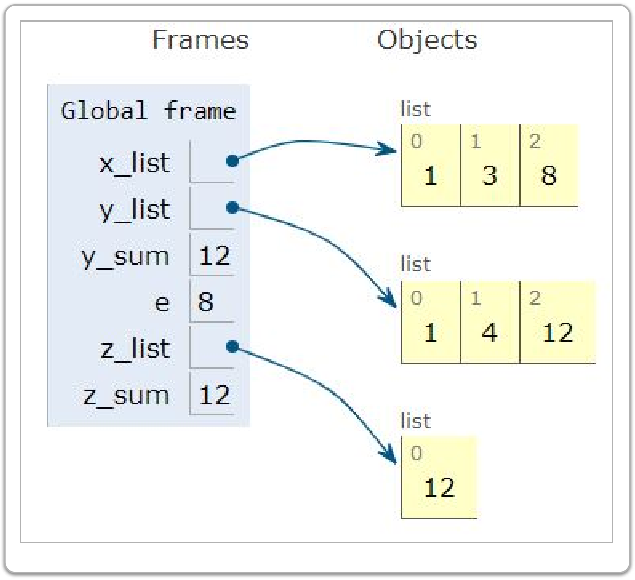
Comparing the different indentation levels in the program below. From the output results, we can see that:

- all sums for y_list
- only one sum for z_list

```
# list of cumulative sums
x_list = [1, 3, 8]

y_list = []
y_sum  = 0
for e in x_list :
    y_sum = y_sum + e
    y_list.append(y_sum)

z_list = []
```

Processing math: 100%

```
for e in x_list :
        z_sum = z_sum + e
    z_list.append(z_sum)
```



Test Yourself Exercises

Test Yourself 1.08

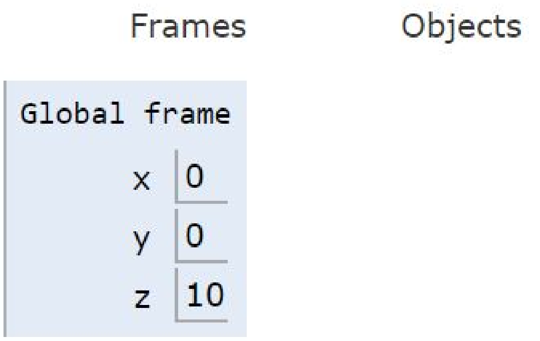
What is the output of A?

```
x = 0
y = 0
z = 10
if z > 25:
    x = z**2
    y = z**3
print(x, y)
```

▶ Show Hint

Suggested output:

```
0 0
```



Test Yourself 1.09

Processing math: 100%

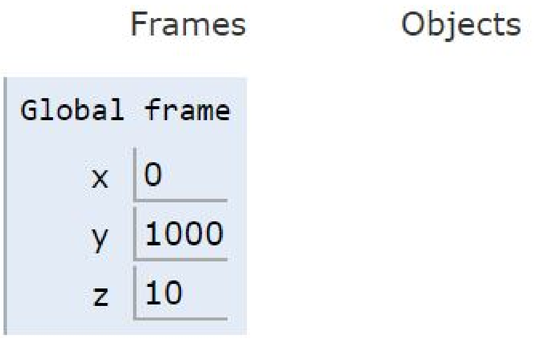
What is the output of B?

```
x = 0
y = 0
z = 10
if z > 25:
    x = z**2
y = z**3
print(x, y)
```

▶ Show Hint

Suggested output:

```
0 1000
```



x	0
y	1000
z	10

Python Modules

- Large Python programs can be split into "modules".
- Modules include variables and functions.
- A module is stored as a file.
- A Python script can import:
  - individual programs
  - all programs in a module

Module Examples

Recall the "compute_area.py" program we used earlier. It's displayed below:

```
def compute_area(x):
    area = x * x
    return area

input_str = input('Enter side of a square: ')
if input_str.isnumeric () is True:
     side = float(input_str)
     area = side * side
     print('area of the square: ', area)
else:
     print ('cannot cast ', input_str)
```

Processing math: 100% ute area.py" into:

1. "helper functions.py" module

```
def compute_area(x):
    area = x * x
    return area
```

2. "main program.py" module

```
input_str = input('Enter side of a square: ')

if input_str.isnumeric () is True:
     side = float(input_str)
     area = side * side
     print('area of the square: ', area)
else:
     print ('cannot cast ', input_str)
```

Call a Function Defined in Another Module

How to call a function defined in another module?

1. Import specific function(s).

   example: "main_program.py" module

```
from helper_functions import compute_area

input_str = input('Enter side of a square: ')

if input_str.isnumeric () is True:
     side = float(input_str)
     area = compute_area(side)
     print('area of the square: ', area)
else:
     print ('cannot cast ', input_str)
```

   Usage: to use this function, refer to the function name.

2. Import complete module.

   Example: "main_program.py" module

```
import helper_functions

input_str = input('Enter side of a square: ')

if input_str.isnumeric () is True:
     side = float(input_str)
     area = helper_functions.compute_area(side)
     print('area of the square: ', area)
else:
     print ('cannot cast ', input_str)
```

   Usage: to use this function, refer to "module.function".

3. Import complete module, using shorter name.

   Example: "main_program.py" module

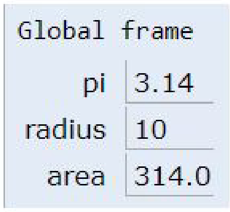
```
import helper_functions as hlp

input_str = input('Enter side of a square: ')

if input_str.isnumeric () is True:
     side = float(input_str)
     area = hlp.compute_area(side)
     print('area of the square: ', area)

     print ('cannot cast ', input_str)
```

Processing math: 100%

Usage: to use this function, refer to "module.function".

Avoiding Ambiguity

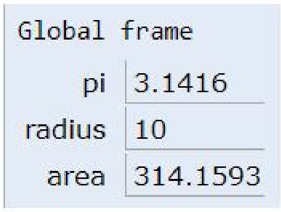
```
from math import pi
pi = 3.14
radius = 10
area = pi * radius **2
```



Global frame

pi	3.14
radius	10
area	314.0

```
pi = 3.14
from math import pi
radius = 10
area = pi * radius **2
```



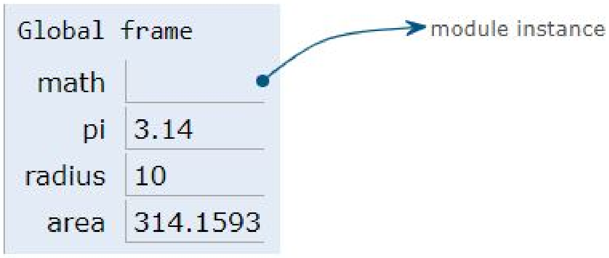
Global frame

pi	3.1416
radius	10
area	314.1593

```
import math
pi = 3.14
radius = 10
area = math.pi * radius **2
```

Same result for:

```
pi = 3.14
import math
radius = 10
area = math.pi * radius **2
```



Global frame

math	
pi	3.14
radius	10
area	314.1593

Preferred Solution

```
import module_a as a
import module_b as b

x = a.function_name()
y = b.function_name()
```

The preferred solution can distinguish functions:

1. with same names
2. and in different modules

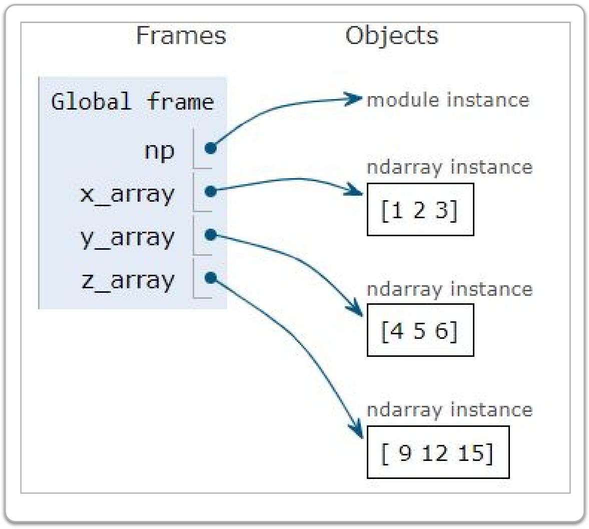
Importing Modules

- There are many modules available to import.
- new objects from basic types.
- Some widely used modules:
    1. numpy (numeric python)
    2. cipy (scientific python)

3. pandas (panel data)
4. matplotlib (plotting)

Example: Numpy

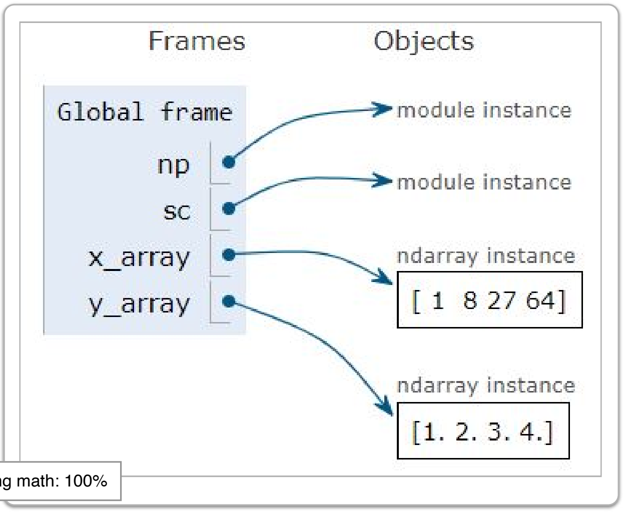
```
import numpy as np

x_array = np.array([1,2,3])
y_array = np.array([4,5,6])
z_array = x_array + 2 * y_array
```



The example uses Numpy for vectorized computations.

Example: Scipy

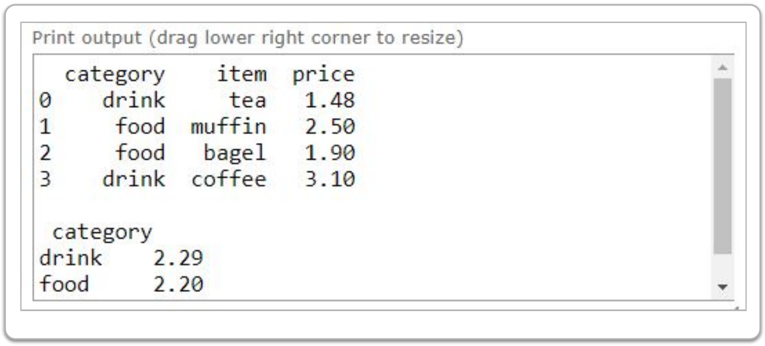
```
import numpy as np
import scipy as sc

# compute cubic roots
x_array = np.array([1,8, 27,64])
y_array = sc.cbrt(x_array)
```



The example uses Scipy and is built on top on Numpy.

Example: Pandas

```
import pandas as pd

df = pd.DataFrame( {
   'category': ['drink','food','food','drink'],
   'item': ['tea','muffin','bagel','coffee'],
   'price': [1.48, 2.50, 1.90, 3.10]},
    columns = ['category','item','price'] )

df_aver=df.groupby(['category'])['price'].mean()
print(df, '\n', df_aver)
```



```
  category    item  price
0    drink     tea   1.48
1     food  muffin   2.50
2     food   bagel   1.90
3    drink  coffee   3.10

 category
drink     2.29
food      2.20
```

Dataframes are similar to tables.

Summary

- A Python program consists of statements.
- No explicit declaration is necessary.
- Can be run in interactive ("shell") or script mode.
- Code blocks are identified by indentation.
- Large programs are split into modules.
- Modules or individual functions can be imported.

Boston University Metropolitan College