Python CS-521

Eugene Pinsky
Department of Computer Science
Metropolitan College, Boston University
Boston, MA 02215
email: epinsky@bu.edu

April 22, 2020

Abstract

This course will present an effective approach to help you learn Python. With extensive use of graphical illustrations, we will build understanding of Python and its capabilities by learning through many simple examples and analogies. The class will involve active student participation, discussions, and programming exercises. This approach will help you build a strong foundation in Python that you will be able to effectively apply in real-job situations and future courses.

INTRODUCTION

Overview:

- understand a basic structure of a Python program
- learn to perform casting
- compare shell and script mode for running Python programs
- learn how to use input function
- learn to perform casting
- learn Python conventions and syntax
- understand Python indentation
- distinguish global and local scope
- learn mudules in Python

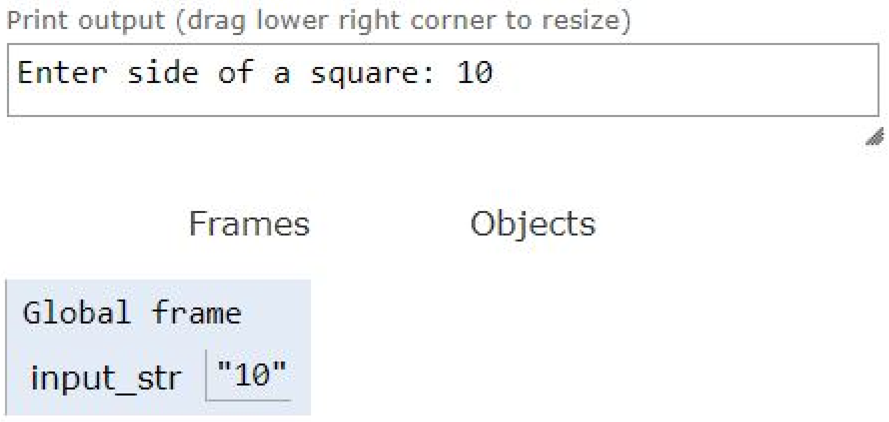
A Simple Program

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

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

A Simple Program (cont'd)

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



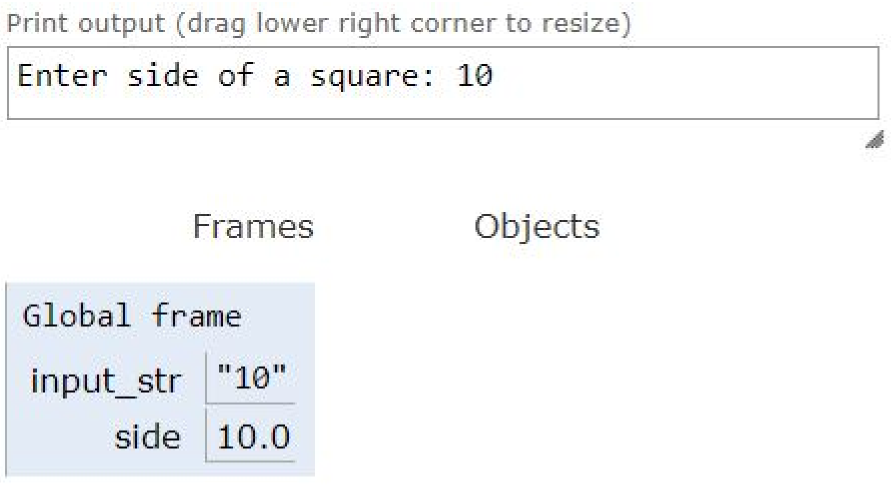
```
Enter side of a square: 10
```

Global frame
input_str "10"

- input is a string

A Simple Program (cont'd)

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



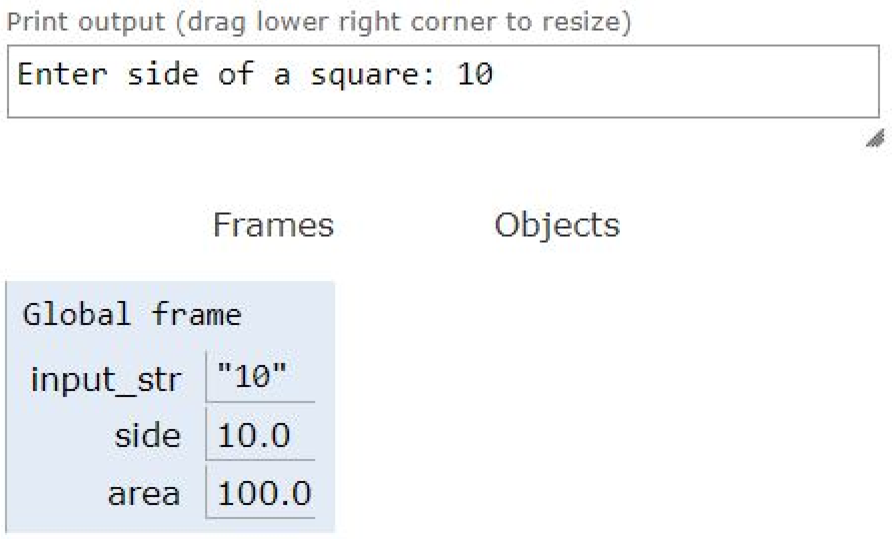
```
Enter side of a square: 10
```

input_str	"10"
side	10.0

• need to ”cast” into numeric

A Simple Program (cont'd)

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



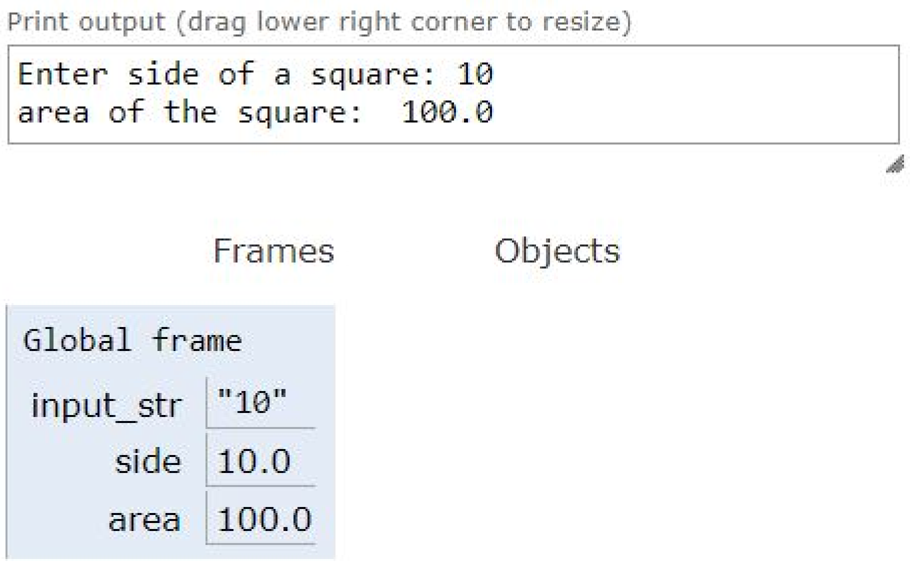
```
Enter side of a square: 10
```

```
input_str "10"
     side 10.0
     area 100.0
```

- computation of area

A Simple Program (cont'd)

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



```
Enter side of a square: 10
area of the square:  100.0
```

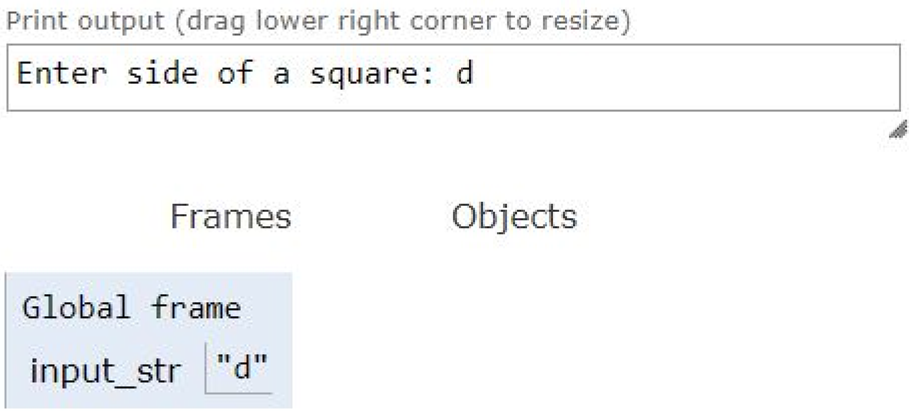
Global frame

input_str	"10"
side	10.0
area	100.0

- output of results

A Simple Program with Errors

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



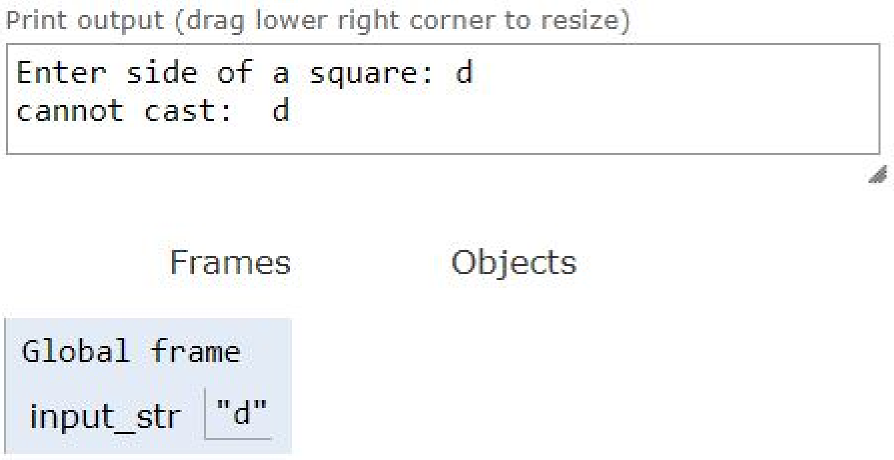
```
Enter side of a square: d
```

```
Global frame
input_str "d"
```

- cannot cast into numeric
- Python will terminate

A Simple Program without Errors

```
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)
```



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

- check if casting is possible

Exercise(s):

- write a program that asks for the side of a cube and computes its volume
- write a program that asks for input and prints it three times
- write a program thet converts temperature in C Farhenheit to temperature in C in Celsius:

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

Python Program Structure

- interpreted, high-level language
- programs have extension *.py*
- two ways to run a program
  1. shell mode
  2. script mode

Shell Mode

```
(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
>>>
```

- ”interactive” use

Script Mode

- code in "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)
```

- run 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

- English: sentences
  1. building blocks (nouns, verbs, adjectives, adverbs)
  2. grammar (rules)
- Python: statements
  1. data types (integers, strings, lists, sets, user-defined classes)
  2. syntax (rules)

Conventions and Syntax

- program contain 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

- building blocks in a language
- similar to noun, verb
- Python has two groups of types
  1. primitive types (”atoms”)
  2. collections (”molecules”)
- additional special types:
  1. None type
  2. range type

Variable Names

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

```
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

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			

- do not use as identifiers
- mostly lower case

Exercise(s):

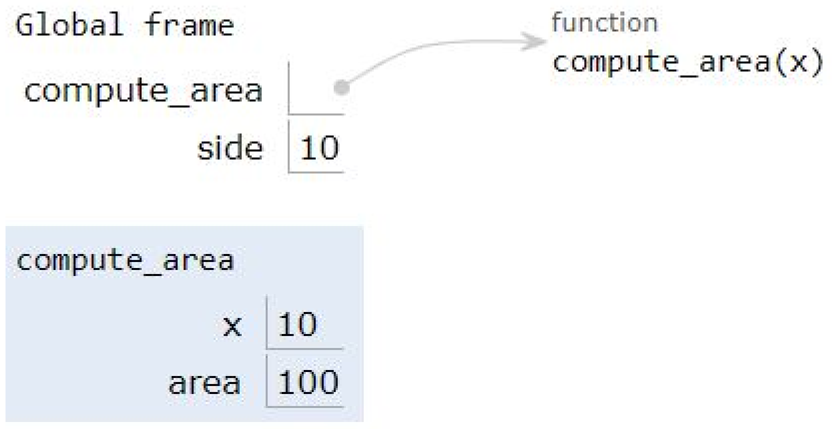
- which of the following are not legal identifiers

```
a    = 5
a2   = 5
a 2  = 5
a_2  = 5
_a_2 = 5
a-5  = 5
a$2  = 5
a2$  = 5
$a2  = 5
```

Local Scope

- a variable inside a function

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

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

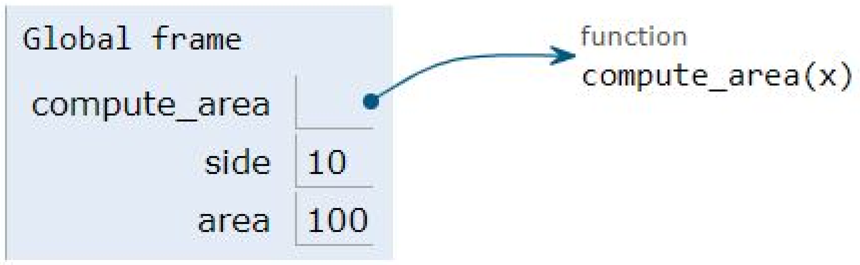


- variable ”area” in *compute_area*() function has local scope

Global Scope

- a variable in the main program body

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

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



- variables ”side” & ”area” have global scope

Exercise(s):

- what is the output of A:

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

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

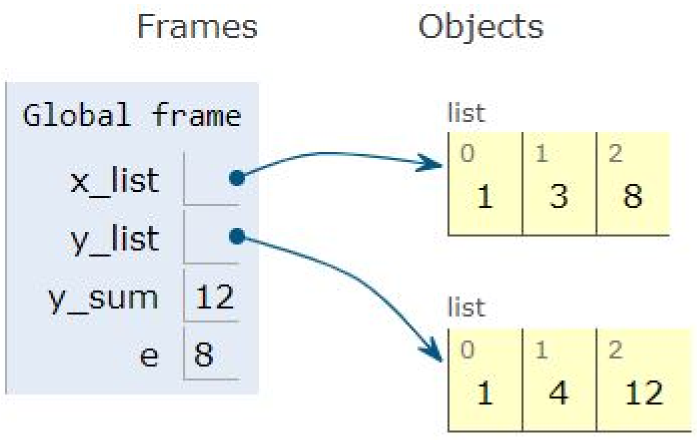
- what is the output of B:

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

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

- are the program outputs the same? Why?

Indentation

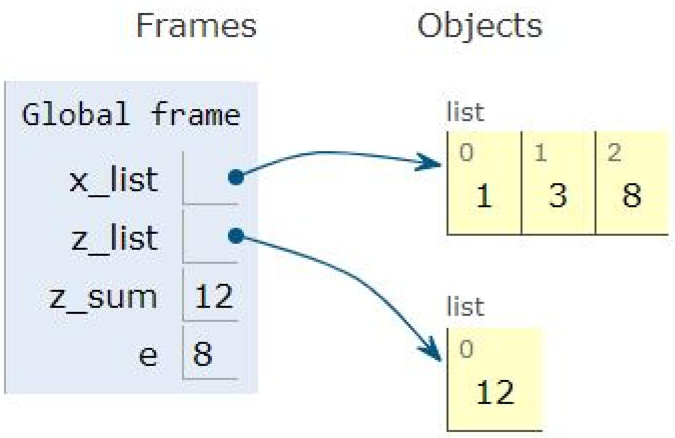
```
# list of cumulative sums
x_list = [1, 3, 8]

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



- all sums computed

Indentation (cont'd)

```
# 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)
```

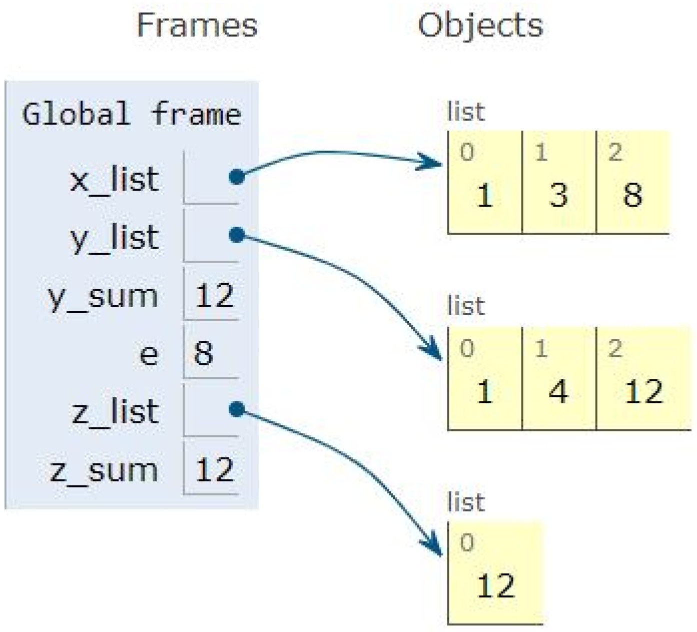


- only one sum is computed

Indentation Comparison

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

Indentation Comparison



- all sums for y_list
- only one sum for z_list

Exercise(s):

- what is the output of A:

```
x = 0
y = 0
z = 10

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

- what is the output of B:

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

Python Modules

- large programs can be split into ”modules”
- variables and functions
- a module is stored as a file
- a Python script can import:
  1. individual programs
  2. all programs in a module

Module Example

- split "compute_area.py" into:

1. "helper_functions.py"

```
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 = compute_area(side)
    print('area of the square: ', area)
else:
    print('cannot cast ', input_str)
```

Module Example

- how to call a function defined in another module?
- import specific function(s)
- ”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: same function name

Module Example (cont'd)

- import complete module
- "main_program.py" module:

```python
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: "module.function"

Module Example (cont'd)

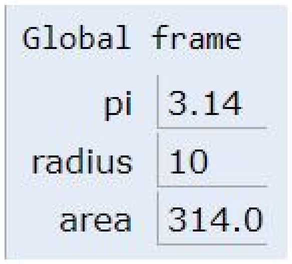
- import complete module
- use shorter name
- "main_program.py" module:

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

- usage: "short_name.function"

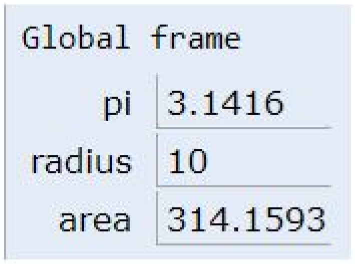
Avoiding Ambiguity

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



```
Global frame
    pi  3.14
radius  10
  area  314.0
```

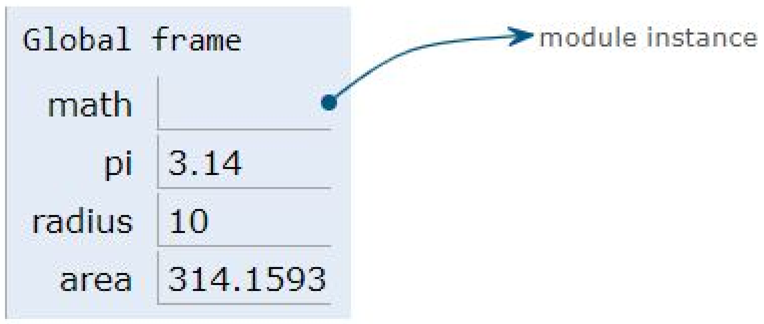
```
pi = 3.14
from math import pi
radius = 10
area = pi * radius**2
```



```
Global frame
    pi  3.1416
radius  10
  area  314.1593
```

Avoiding Ambiguity (cont'd)

```
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
```

Avoiding Ambiguity (cont'd)

- preferred solution

```python
import module_a as a
import module_b as b

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

- can distinguish functions

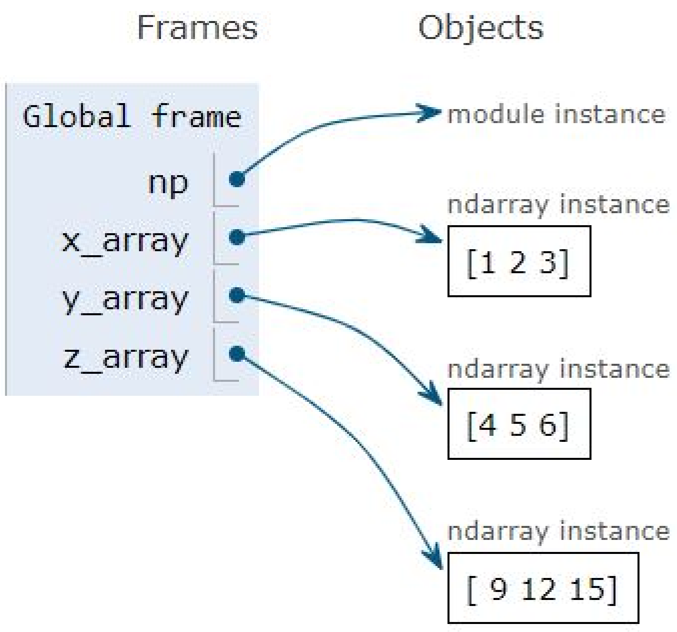
1. with same names
2. and in different modules

Importing Modules

- many modules available
- new objects from basic types
- some widely used modules:
  1. numpy (numeric python)
  2. scipy (scientific python)
  3. pandas (panel data)
  4. matplotlib (plotting)

Example: Numpy

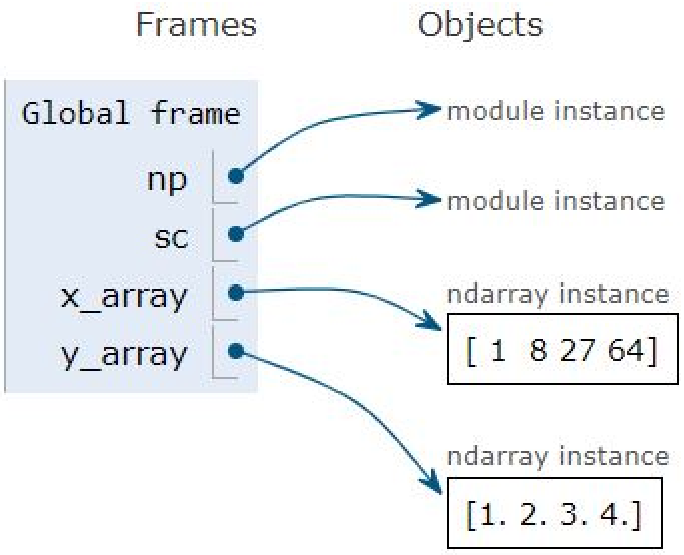
```
import numpy as np

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



- vectorized computations

Example: Scipy

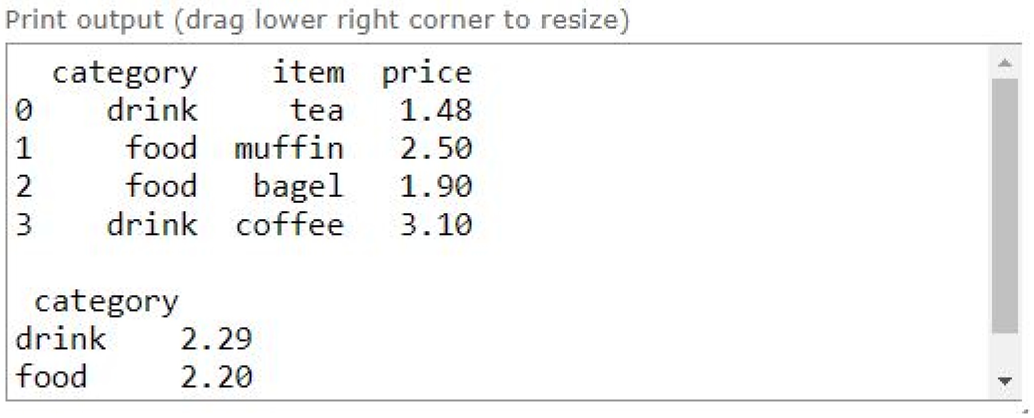
```
import numpy as np
import scipy as sc

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



- 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)
```


Print output (drag lower right corner to resize)

```
  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 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