METU Computer Engineering

CEng 240 – Spring 2021
Week 4

Sinan Kalkan

Dive into Python [Part 2]

Disclaimer: Figures without reference are from either from “Introduction to programming concepts with case studies in Python” or “Programming with Python for Engineers”, which are both co-authored by me.

Previously on CENG240!

Basic Data in Python
Numerical Types

- Integers:
  - `int`
  - Unlimited size
- Floating point numbers:
  - `float`
  - IEEE754 standard (32bit, 64bit)
- Complex numbers
  - `complex`
  - `3+4j`

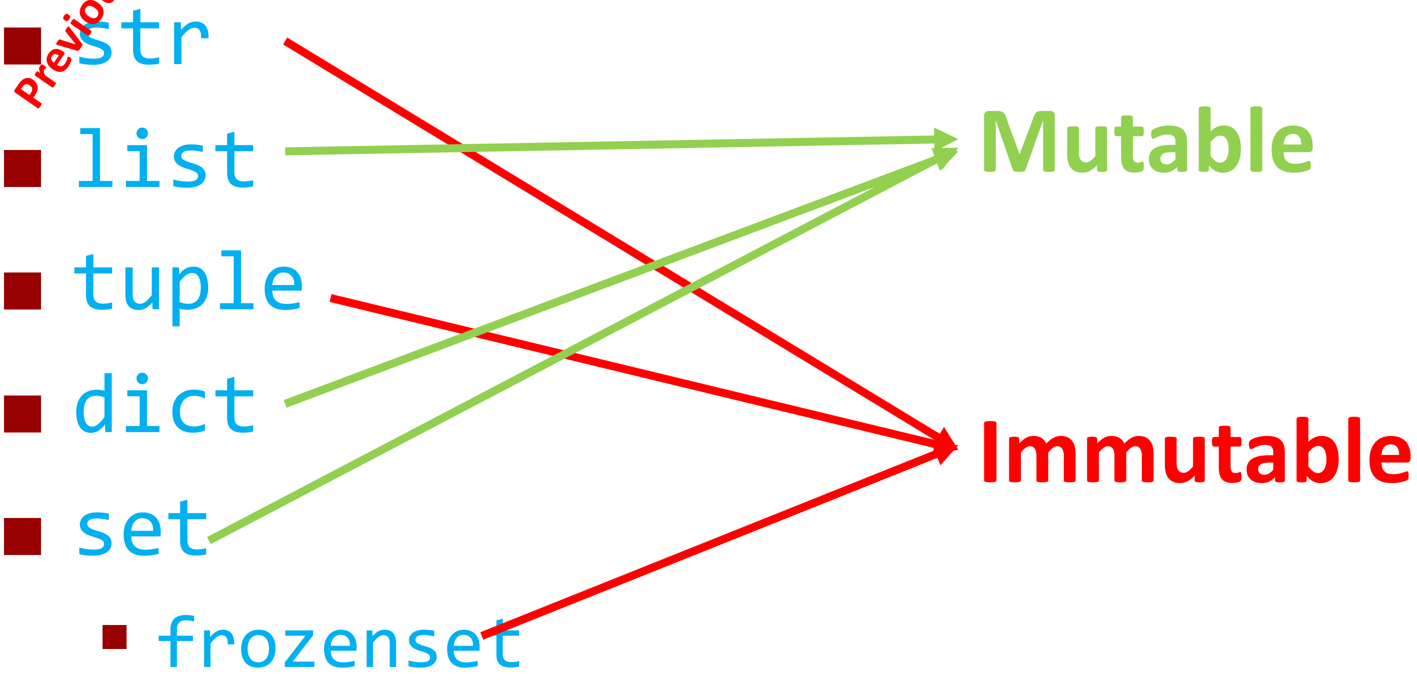
Previously on CENG240!

Basic Data in Python
Boolean Type

- `bool` type
  - Can take `True` or `False`
- Useful operations with `bool` type
  - `and, or, not`

Previously on CENG240!

Container Data in Python



- `str`
- `list`
- `tuple`
- `dict`
- `set`
    - `frozenset`

Mutable

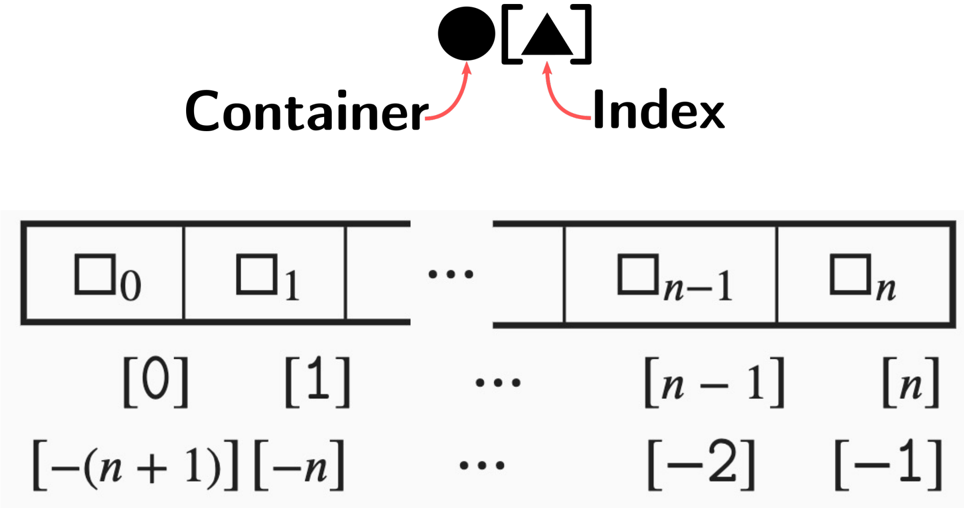
Immutable

- Mutability vs. immutability

Container Data in Python

Accessing Elements of Sequences

- Positive indexing
- Negative indexing
- Slicing

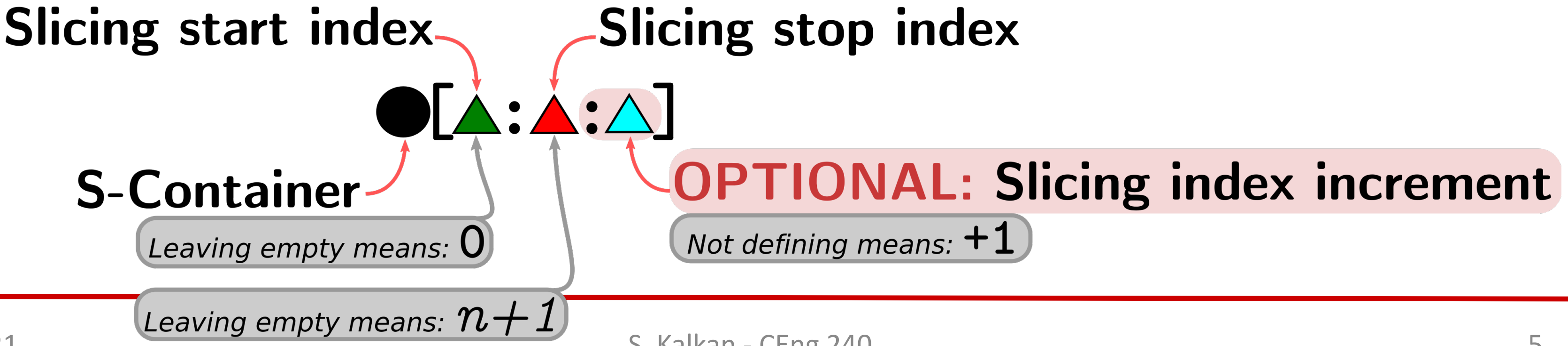


●[▲]

Container

Index

$\square_0$	$\square_1$	⋯	$\square_{n-1}$	$\square_n$
[0]	[1]	⋯	$[n-1]$	$[n]$
$[-(n+1)]$	$[-n]$	⋯	[−2]	[−1]



Slicing start index

Slicing stop index

●[▲:▲:▲]

S-Container

Leaving empty means: 0

Leaving empty means: $n+1$

OPTIONAL: Slicing index increment

Not defining means: +1

Previously on CENG240!

Action

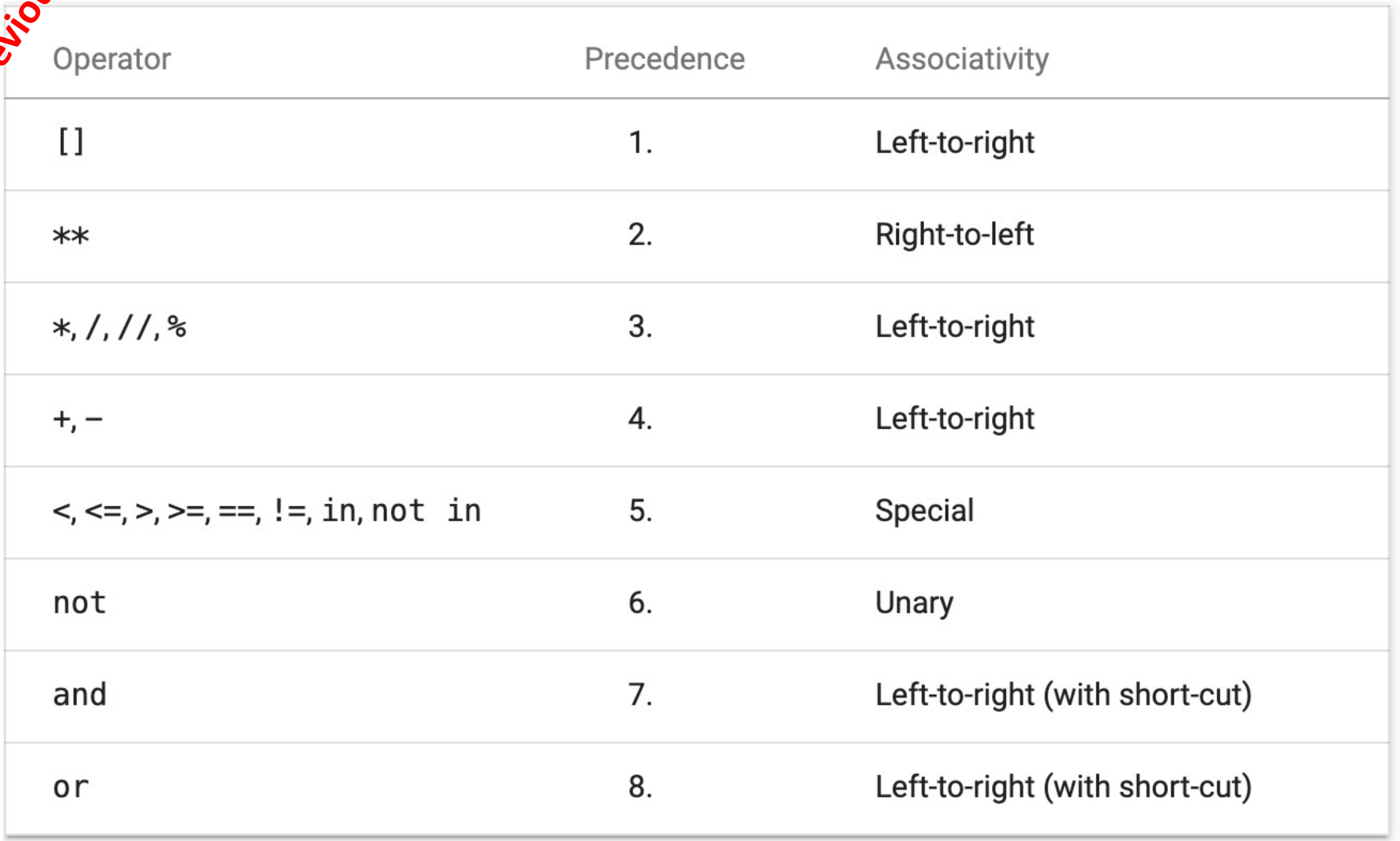
- Purposes of actions
  - Creating/modifying data
  - Interaction with the environment

- Types of actions
  - Expressions
  - Statements

Previously on CENG240!

Expression Evaluation

Precedence and Associativity



Operator	Precedence	Associativity
[]	1.	Left-to-right
**	2.	Right-to-left
*, /, //, %	3.	Left-to-right
+, −	4.	Left-to-right
<, <=, >, >=, ==, !=, in, not in	5.	Special
not	6.	Unary
and	7.	Left-to-right (with short-cut)
or	8.	Left-to-right (with short-cut)

Previously on CENG240!

Statements

- Basic statements
  - `del L[3]`
  - `a = 20`
  - `pass, del, return, yield, raise, break, continue, import, future, global, nonlocal.`
- Compound statements
  - Conditional statement
  - Repetition statements

Previously on CENG240!

Statements: Assignment

METU Computer Engineering

- Simple assignment
  - `a = 10`
- Multiple assignment
  - `a = b = c = 10`
  - `a, b = 10, 20`
- Compound assignment
  - `a += 10`
  - `a *= 20`
- Swapping values
  - `a, b = b, a`

`id()` function

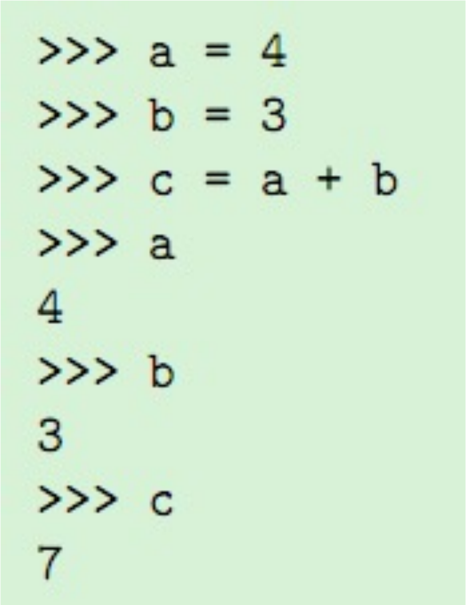
This Week

- Dive into Python [Part 2/2]
  - Variables; Aliasing problem; Naming Variables
  - Actions for interacting with the environment
  - Actions that are ignored (comments, pass statements)
  - Actions in packages (libraries)
  - Writing your actions (interpreter vs. script/modules)

Administrative Notes

- Quiz 3 announced!
- Labs started
- Midterm: 1 June, Tuesday, 17:40

Variables in Python

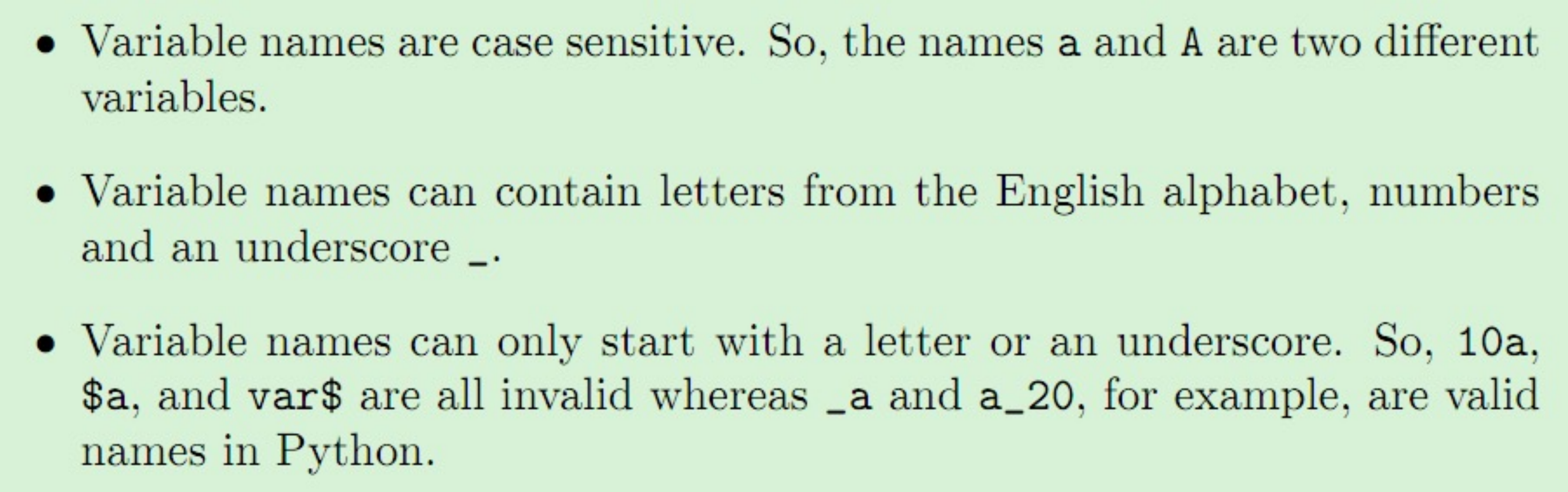


```
>>> a = 4
>>> b = 3
>>> c = a + b
>>> a
4
>>> b
3
>>> c
7
```

- We don’t need to define a variable before using it.
- We don’t need to specify the type of a variable.

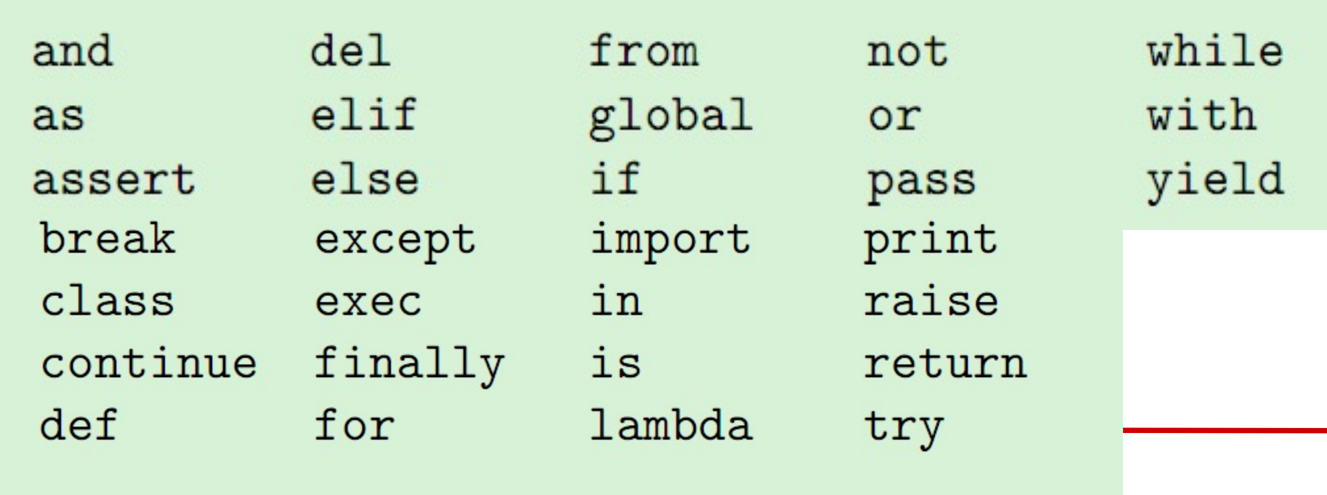
- ‘=‘ means “Change the content of the variable with the value at the right-hand side”.
  - Assignment!
- The left-side of the assignment should be a valid variable name:
  - Ex: a+2 = 5 → NOT VALID!

Variable Naming in Python



- Variable names are case sensitive. So, the names a and A are two different variables.
- Variable names can contain letters from the English alphabet, numbers and an underscore _.
- Variable names can only start with a letter or an underscore. So, 10a, $a, and var$ are all invalid whereas _a and a_20, for example, are valid names in Python.

- Variable names cannot be one of the keywords in Python:

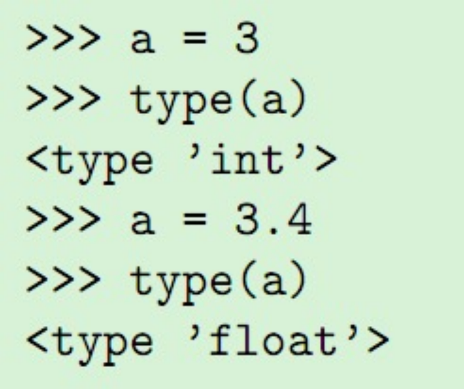


```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

METU Computer Engineering

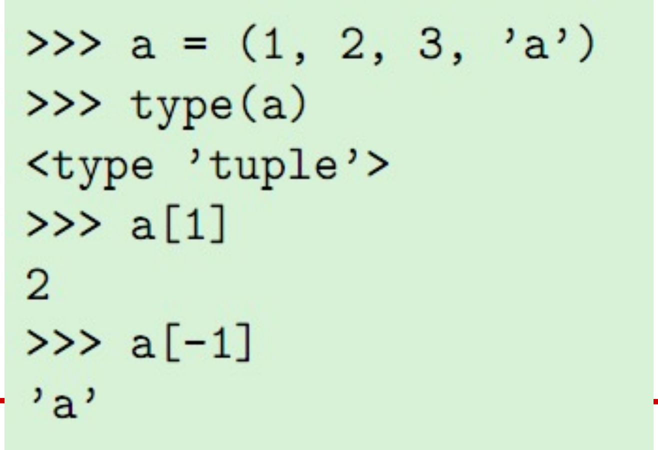
More on Variables in Python

- Typing of variables:
  - Python is dynamically typed:



```
>>> a = 3
>>> type(a)
<type ’int’>
>>> a = 3.4
>>> type(a)
<type ’float’>
```

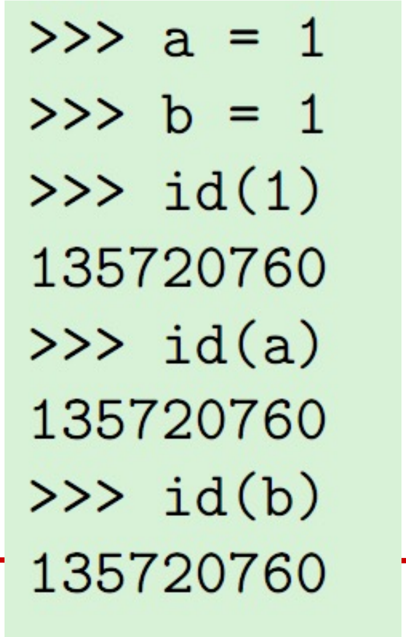
- Using variables:



```
>>> a = (1, 2, 3, ’a’)
>>> type(a)
<type ’tuple’>
>>> a[1]
2
>>> a[-1]
’a’
```

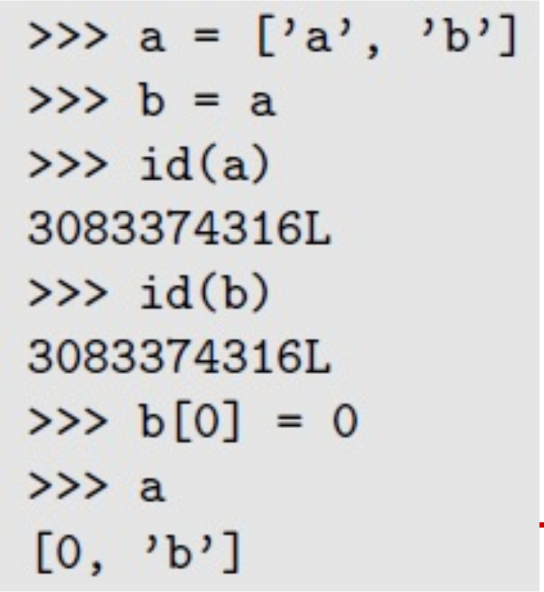
Variables, Values and Aliasing in Python

- Every data (whether constant or not) has an identifier (an integer) in Python:



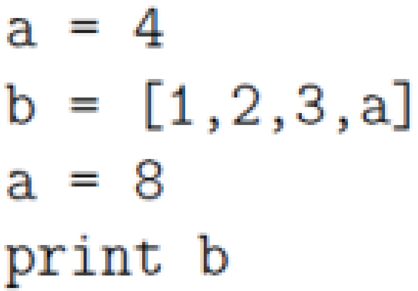
```
>>> a = 1
>>> b = 1
>>> id(1)
135720760
>>> id(a)
135720760
>>> id(b)
135720760
```

- If the type of the data is mutable, there is a problem!!!

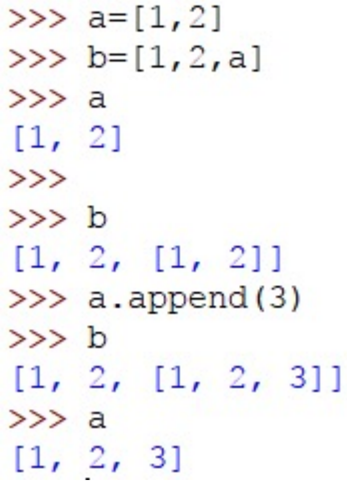


```
>>> a = [’a’, ’b’]
>>> b = a
>>> id(a)
3083374316L
>>> id(b)
3083374316L
>>> b[0] = 0
>>> a
[0, ’b’]
```

This is called Aliasing.

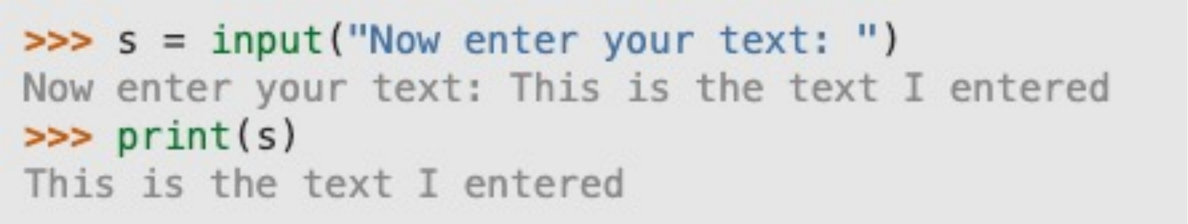


```
a = 4
b = [1,2,3,a]
a = 8
print b
```

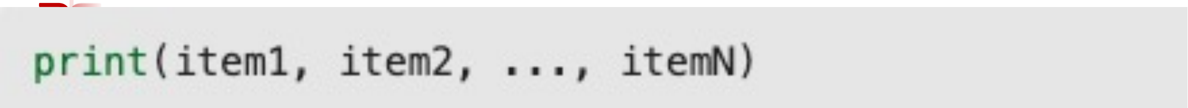


```
>>> a=[1,2]
>>> b=[1,2,a]
>>> a
[1, 2]
>>>
>>> b
[1, 2, [1, 2]]
>>> a.append(3)
>>> b
[1, 2, [1, 2, 3]]
>>> a
[1, 2, 3]
```

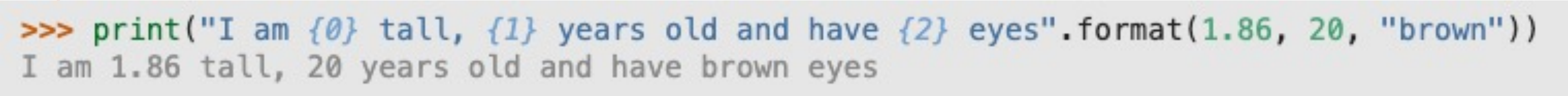
Actions for I/O



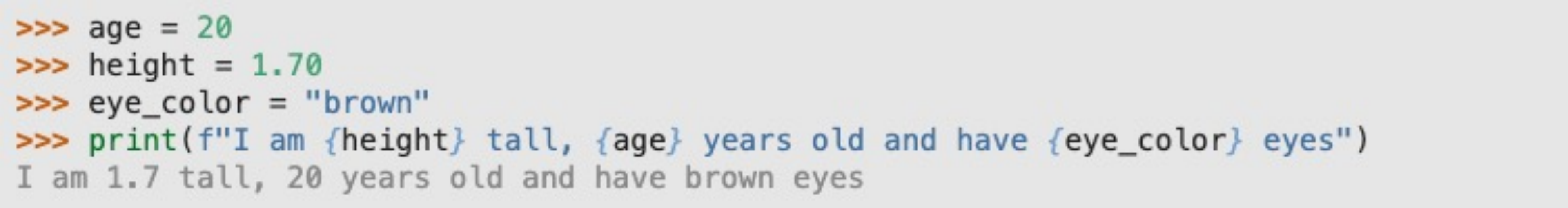
```
>>> s = input("Now enter your text: ")
Now enter your text: This is the text I entered
>>> print(s)
This is the text I entered
```



```
print(item1, item2, ..., itemN)
```



```
>>> print("I am {0} tall, {1} years old and have {2} eyes".format(1.86, 20, "brown"))
I am 1.86 tall, 20 years old and have brown eyes
```

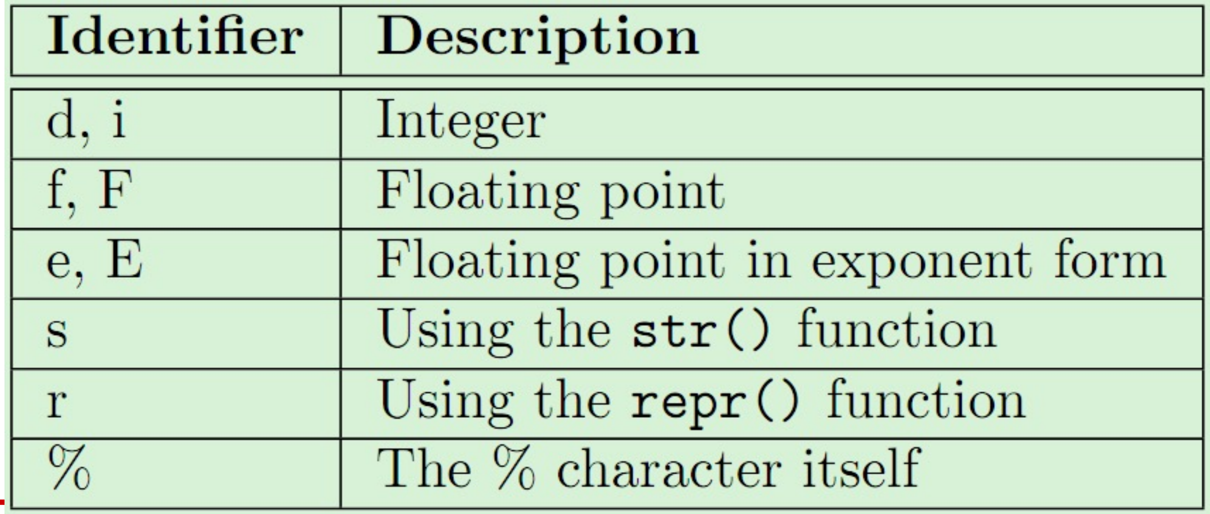


```
>>> age = 20
>>> height = 1.70
>>> eye_color = "brown"
>>> print(f"I am {height} tall, {age} years old and have {eye_color} eyes")
I am 1.7 tall, 20 years old and have brown eyes
```

Actions for I/O

```
>>> print("I am %f tall, %d years old and have %s eyes" % (1.7569, 20, "blue"))
I am 1.756900 tall, 20 years old and have blue eyes

>>> print("I am %.2f tall, %d years old and have %s eyes" % (1.7569, 20, "blue"))
I am 1.76 tall, 20 years old and have blue eyes
```

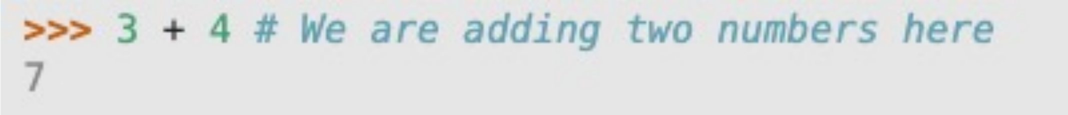
- %f → Data identifier
- We have the following identifiers in Python:



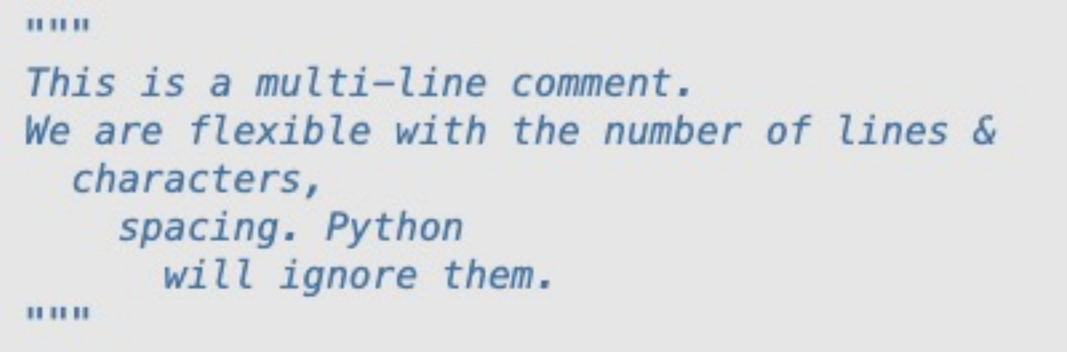
Identifier	Description
d, i	Integer
f, F	Floating point
e, E	Floating point in exponent form
s	Using the `str()` function
r	Using the `repr()` function
%	The % character itself

METU Computer Engineering

Actions that are ignored: Comments



```
>>> 3 + 4 # We are adding two numbers here
7
```

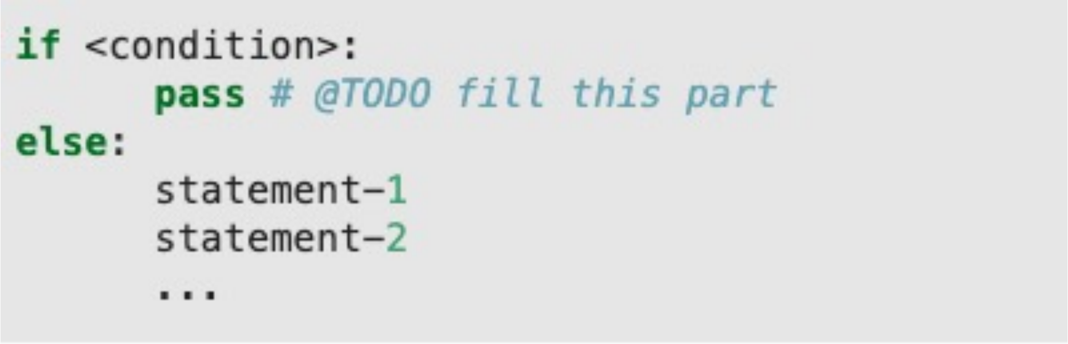


```
"""
This is a multi-line comment.
We are flexible with the number of lines &
 characters,
  spacing. Python
   will ignore them.
"""
```

METU Computer Engineering

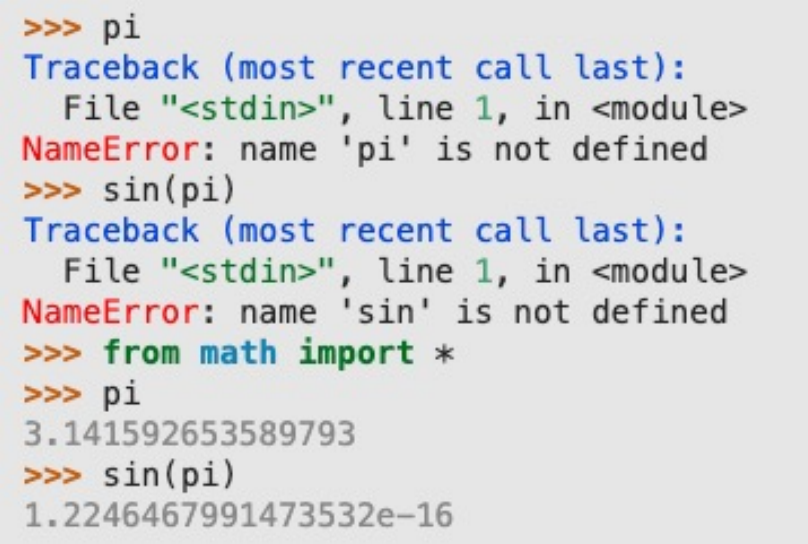
METU Computer Engineering

Actions that are ignored: pass statement

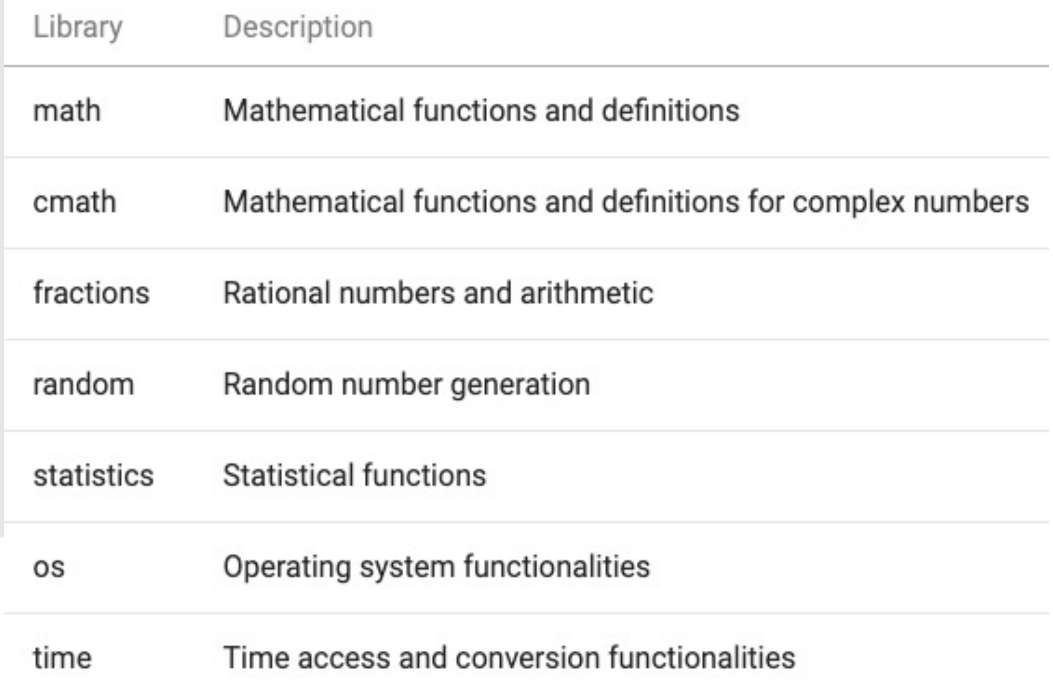


```
if <condition>:
      pass # @TODO fill this part
else:
      statement-1
      statement-2
      ...
```

Actions in packages

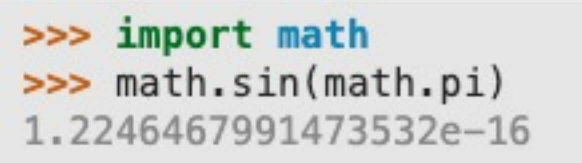


```
>>> pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> sin(pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined
>>> from math import *
>>> pi
3.141592653589793
>>> sin(pi)
1.2246467991473532e-16
```

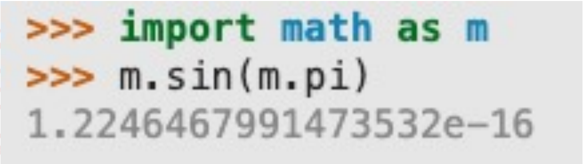


Library	Description
math	Mathematical functions and definitions
cmath	Mathematical functions and definitions for complex numbers
fractions	Rational numbers and arithmetic
random	Random number generation
statistics	Statistical functions
os	Operating system functionalities
time	Time access and conversion functionalities

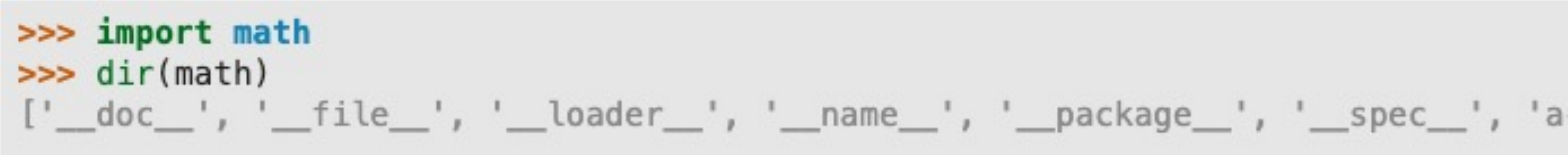
Actions in packages



```
>>> import math
>>> math.sin(math.pi)
1.2246467991473532e-16
```



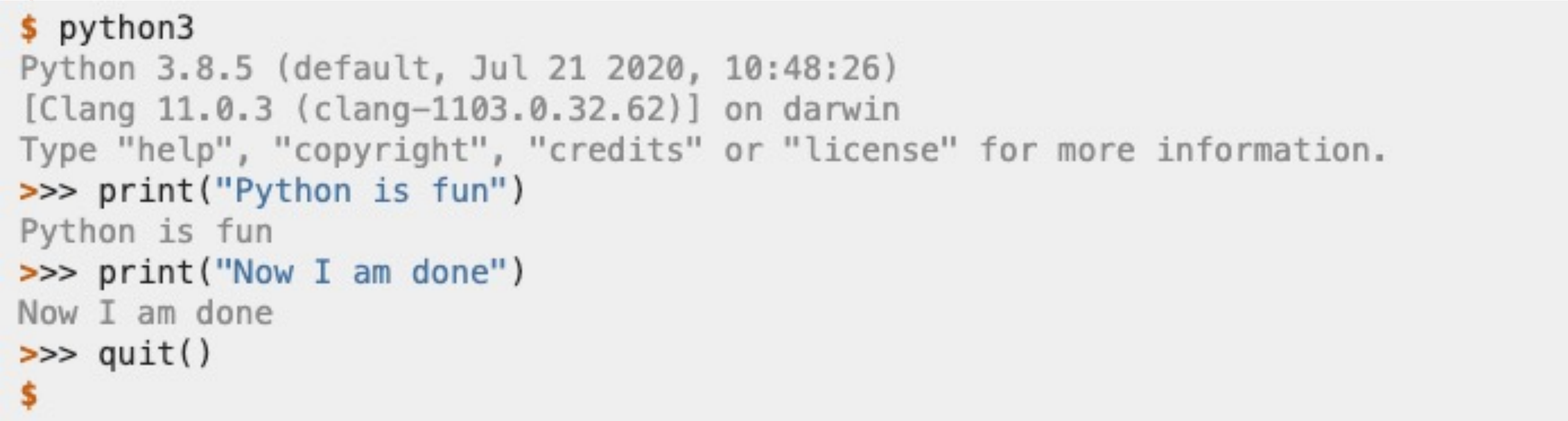
```
>>> import math as m
>>> m.sin(m.pi)
1.2246467991473532e-16
```



```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'a
```

Writing your actions:
(1) Interact with the interpreter

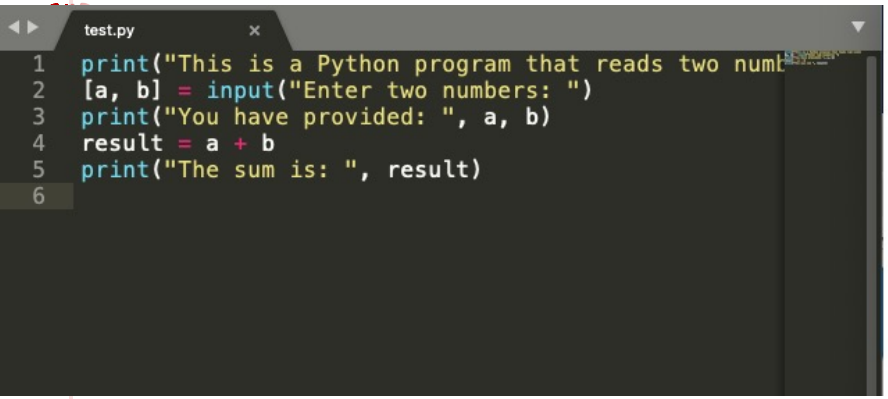
ter Engineering



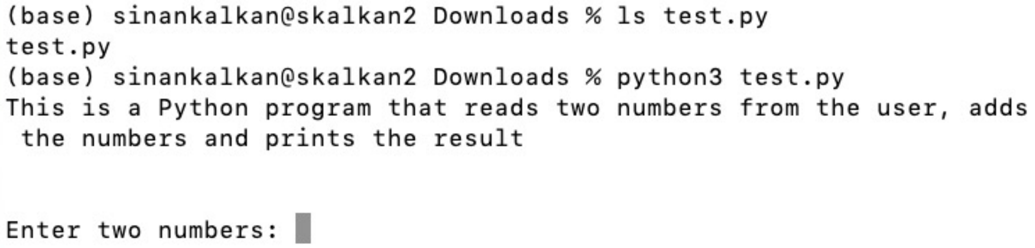
```
$ python3
Python 3.8.5 (default, Jul 21 2020, 10:48:26)
[Clang 11.0.3 (clang-1103.0.32.62)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Python is fun")
Python is fun
>>> print("Now I am done")
Now I am done
>>> quit()
$
```

Writing your actions:

(2) Putting your actions into a script file



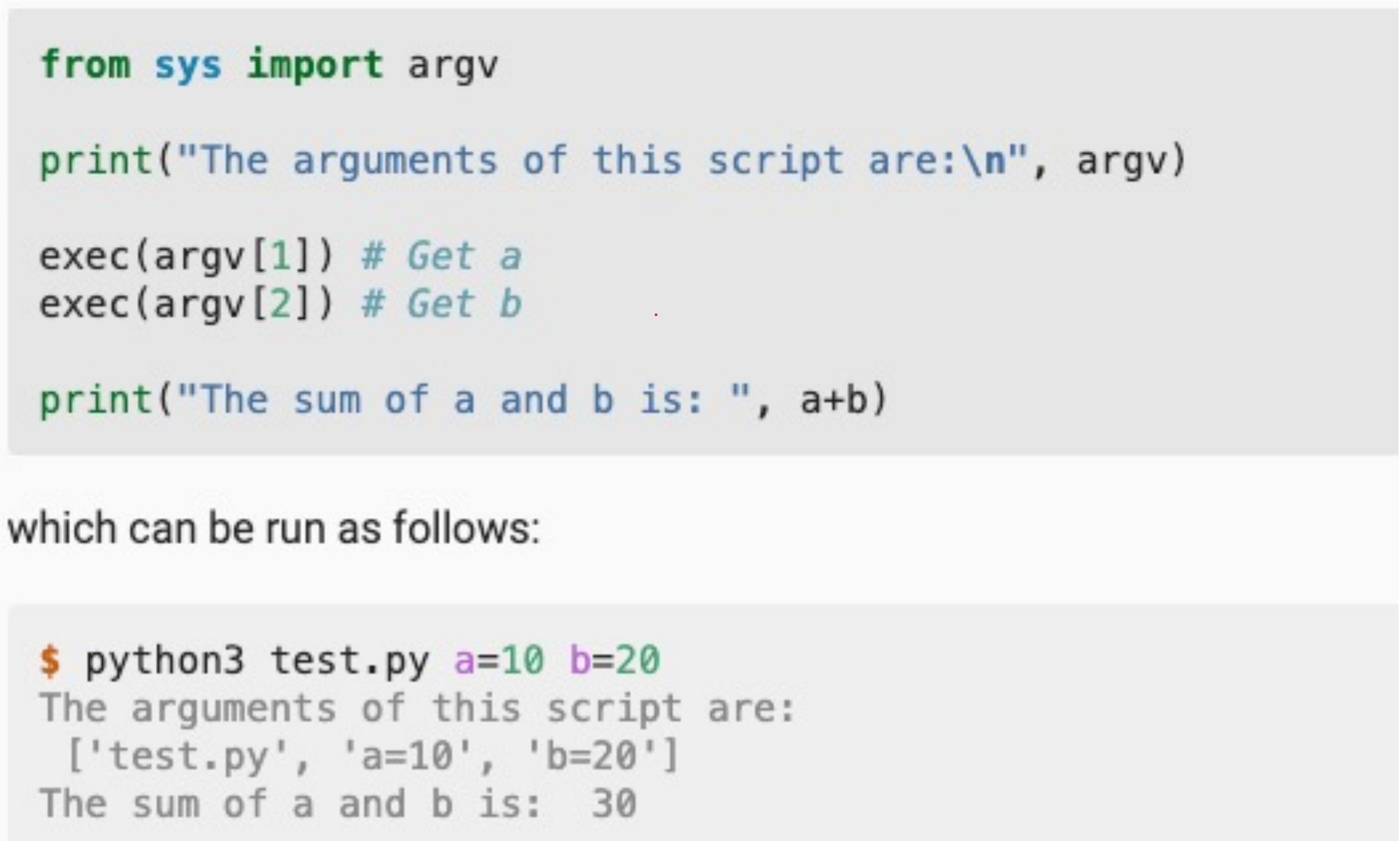
```
print("This is a Python program that reads two numb
[a, b] = input("Enter two numbers: ")
print("You have provided: ", a, b)
result = a + b
print("The sum is: ", result)
```



```
(base) sinankalkan@skalkan2 Downloads % ls test.py
test.py
(base) sinankalkan@skalkan2 Downloads % python3 test.py
This is a Python program that reads two numbers from the user, adds
 the numbers and prints the result


Enter two numbers: 
```

Writing your actions:

(2) Putting your actions into a script file



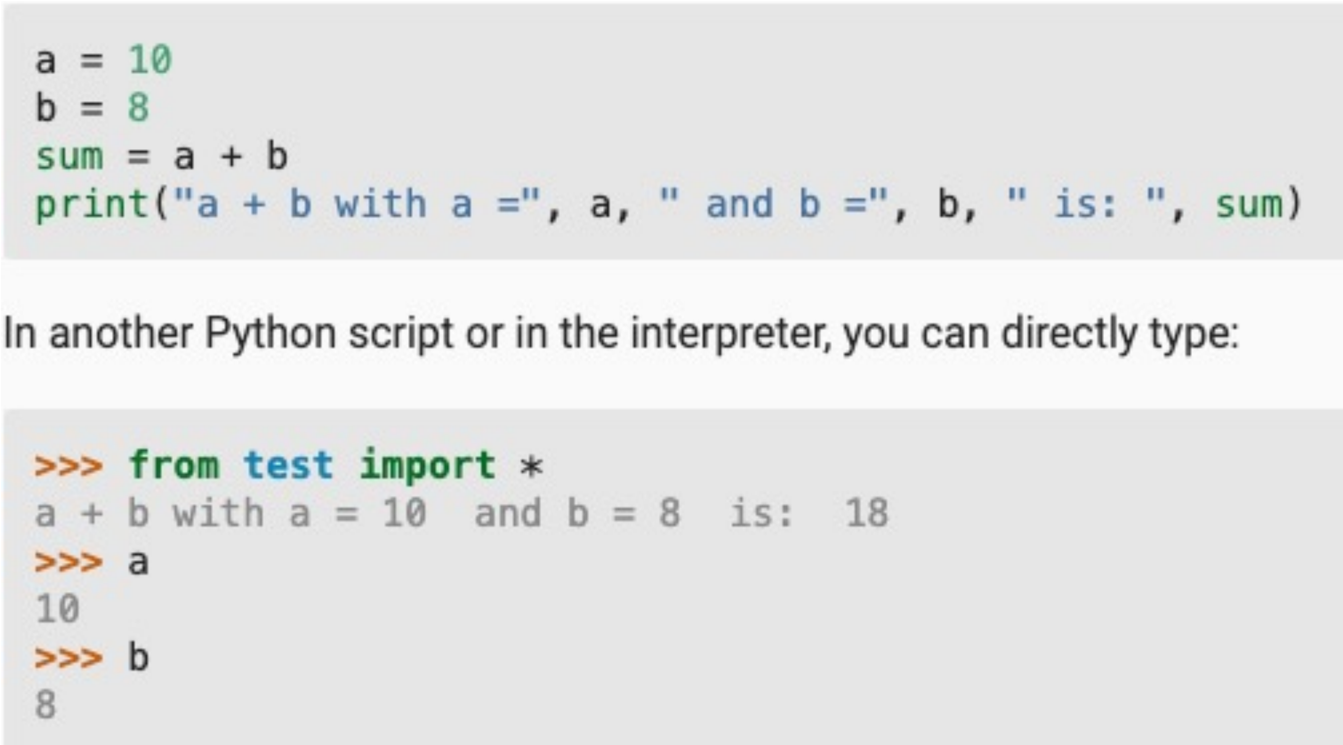
```python
from sys import argv

print("The arguments of this script are:\n", argv)

exec(argv[1]) # Get a
exec(argv[2]) # Get b

print("The sum of a and b is: ", a+b)
```

which can be run as follows:

```
$ python3 test.py a=10 b=20
The arguments of this script are:
 ['test.py', 'a=10', 'b=20']
The sum of a and b is:  30
```

Writing your actions:

(3) Your actions in a module



```
a = 10
b = 8
sum = a + b
print("a + b with a =", a, " and b =", b, " is: ", sum)
```

In another Python script or in the interpreter, you can directly type:

```
>>> from test import *
a + b with a = 10  and b = 8  is:  18
>>> a
10
>>> b
8
```

```
To reload:
>>> from importlib import reload
>>> reload(test)
```

Final Words: Important Concepts

- Variables; Aliasing problem; Naming Variables
- Actions for interacting with the environment
- Comments, pass statements
- Actions in packages (libraries)
- Writing your actions (interpreter vs. script/modules)

THAT’S ALL FOLKS!
STAY HEALTHY