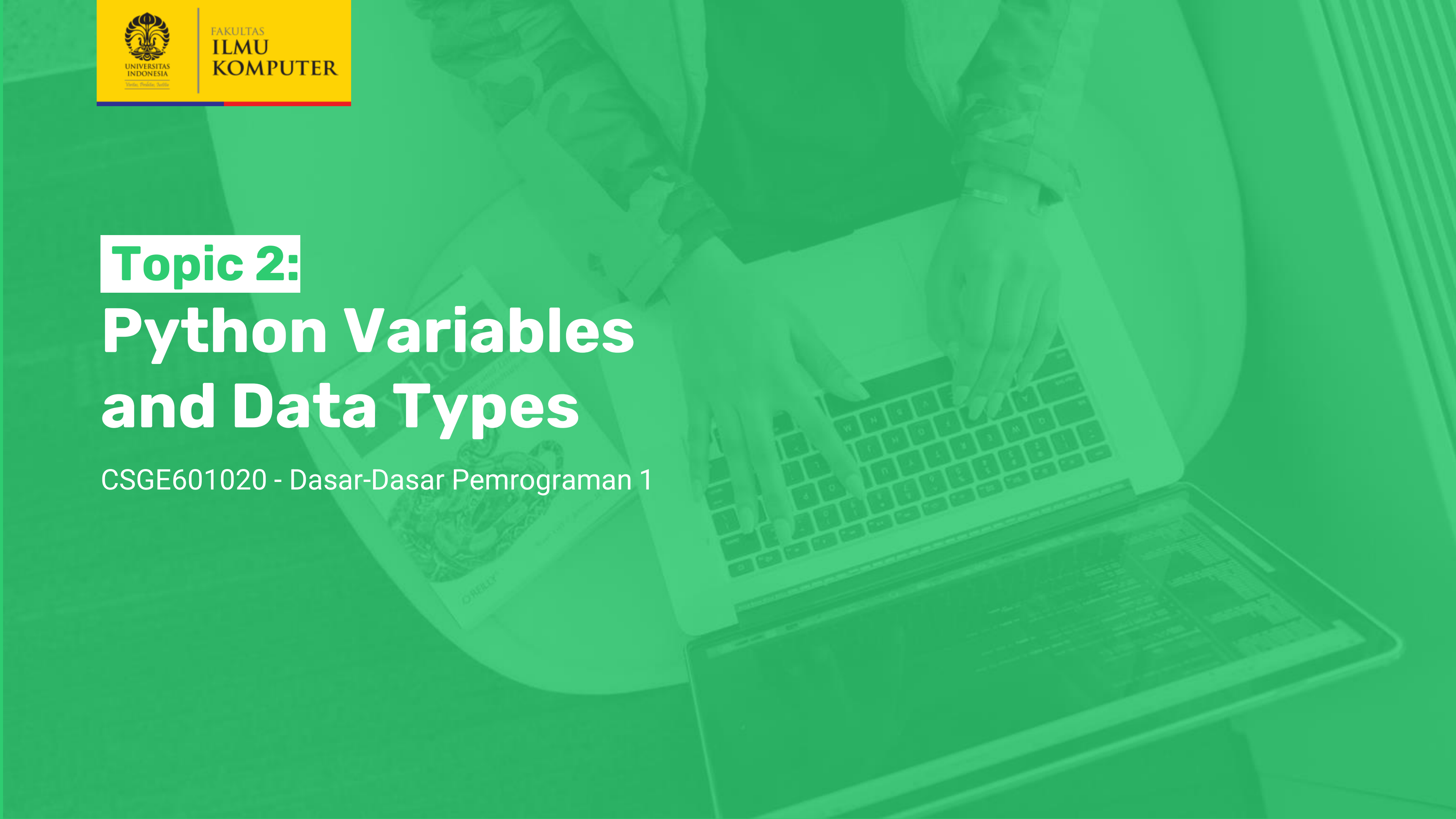
UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER



Topic 2: Python Variables and Data Types

CSGE601020 - Dasar-Dasar Pemrograman 1

Acknowledgement

This slide is an adapted version of ‘Python Variable and Data Types’ slides used in DDP1 Course (2020/2021)
by **Hafizh Rafizal Adnan, M.Kom** and ‘Kuliah 3 Komponen Program dan Variabel’ slides by **Raja O. P. Damanik, M.Sc.**
and **Dr. Eng. Lia Sadita**

Several materials are reused from ‘Variables and Data Types’ slides used in Dasar-Dasar Pemrograman 1 dengan Python (CSGE601020/4 SKS) Course (https://ocw.ui.ac.id/course/view.php?id=142)
by **Fariz Darari, Ph.D.**

Some additional contents, illustrations and visual design elements are provided by **Lintang Matahari Hasani, M.Kom.** *(lintang.matahari01[at]cs.ui.ac.id)*

In this session, you will learn ...

- Module
- Statement vs Expression
- Whitespaces
- Comments
- Python Objects, Variables, and Namespace
- Python Data Type
- Type Conversion
- Python Tokens/Keywords
- Naming conventions

Module

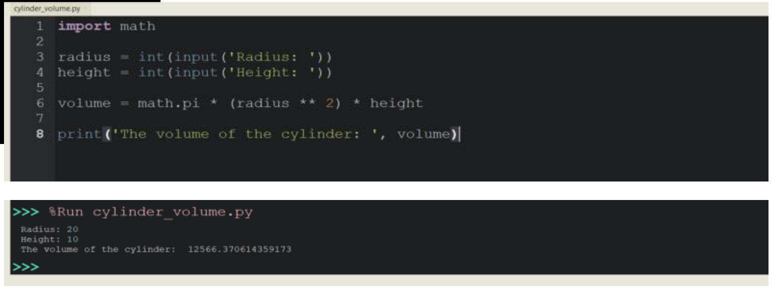
Module is **file** with Python statements.

➔ There are modules provided by Python to perform common tasks (math, database, web interaction, etc.)
➔ The wealth of these modules is one of the great features of Python

More info: https://docs.python.org/3/tutorial/modules.html
List of built-in modules (Python3): https://docs.python.org/3/py-modindex.html

An Example: A Module Named `cylinder_volume.py`

```python
import math

radius = int(input('Radius: '))
height = int(input('Height: '))

volume = math.pi * (radius ** 2) * height

print('The volume of the cylinder: ', volume)
```



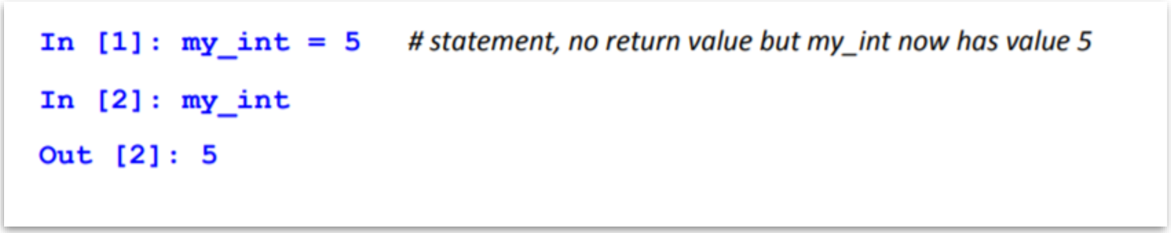
cylinder_volume.py

```python
import math

radius = int(input('Radius: '))
height = int(input('Height: '))

volume = math.pi * (radius ** 2) * height

print('The volume of the cylinder: ', volume)
```

```
>>> %Run cylinder_volume.py
 Radius: 20
 Height: 10
 The volume of the cylinder:  12566.370614359173
>>>
```

Statements

Statements are basic commands in Python.
Statement performs actions, but **does not return any values** when executed.



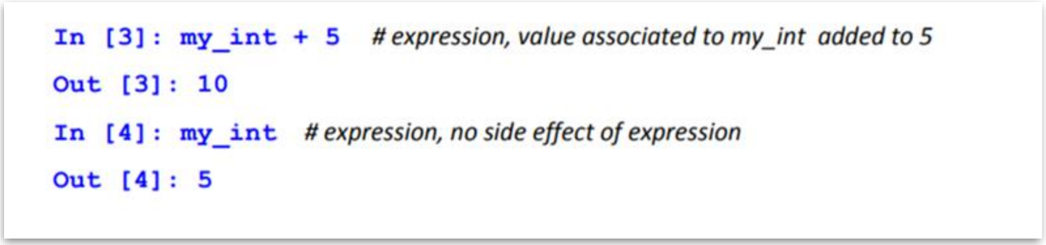
```
In [1]: my_int = 5     # statement, no return value but my_int now has value 5
In [2]: my_int
Out [2]: 5
```

Expressions

Expressions perform some operation and **return a value**.

➔ Expressions can act as statements (evaluate the value of the expression), but statement is not an expression.

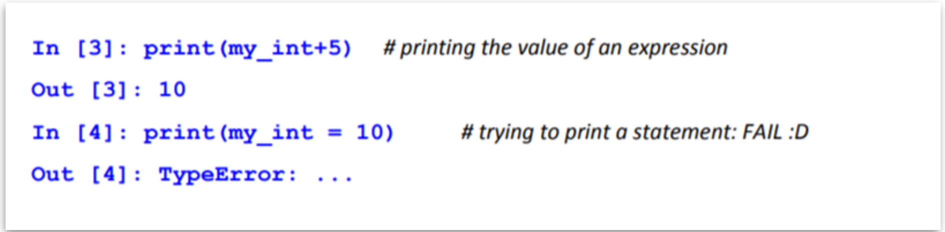
➔ Expressions typically do not modify values in the interpreter



```
In [3]: my_int + 5    # expression, value associated to my_int  added to 5
Out [3]: 10
In [4]: my_int   # expression, no side effect of expression
Out [4]: 5
```

Why Important?

- ➔ Statements are executed (only) and give side effects, but no value given.
- ➔ Expressions are also executed (more precisely, evaluated) and give you a value, which you can print.



```
In [3]: print(my_int+5)     # printing the value of an expression
Out [3]: 10
In [4]: print(my_int = 10)        # trying to print a statement: FAIL :D
Out [4]: TypeError: ...
```

Whitespace

Whitespaces are characters that don't print. The following characters count as whitespace: space, tab, return, linefeed, formfeed, and vertical tab.

➔ Whitespace is ignored within both expressions and statements, **use it to make a program more readable**

```
y=x+5
```

has exactly the same meaning as

```
y          = x + 5
```

➔ But, **leading whitespace** (whitespace at the beginning of a line)—defines indentation — matters. Python requires some consistency in grouping code using indentation. It forces Python programmers manage code’s readibility. The disadvantage is a possibility hassle between number of spaces and tabs that can be different.

➔ Blank lines are also considered to be whitespace.

Long Statement vs Readability: Use Continuation

However, python is **sensitive to white space at the end of line** (new line).

Also, writing long statement
beyond our screen harms our program’s readability.
To make a command continues to the next line, **use the \**

```
print("this is a test", \
"of continuation")
```

Comments

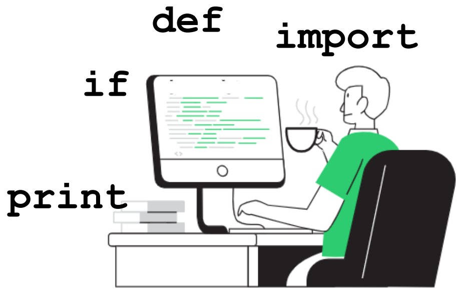
- ➔ A comment begins with a `#` (pound sign)
- ➔ From `#` to the end of that line, nothing will be interpreted by Python (will be ignored).
- ➔ Use comments to write information that will help the reader with the code.

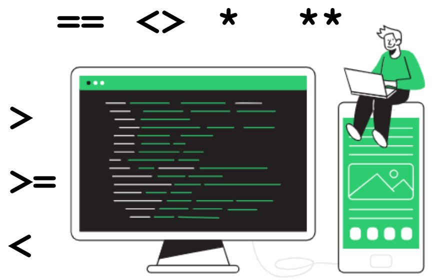
```
# Ini hanya sebuah komentar
# Tuliskan informasi bermakna
```

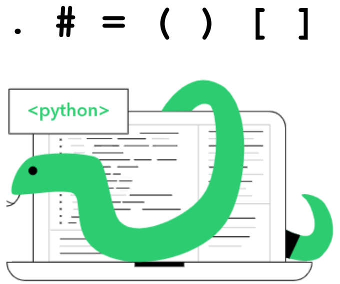
Good comments don't repeat the code or explain it. **They clarify its intent.** Comments should explain, at a higher level of abstraction than the code, what you're trying to do. (Code Complete - McConnell)

If your code contains a novel or noteworthy solution, add comments to **explain the methodology**

Python Tokens







Keywords

Operators

Punctuators

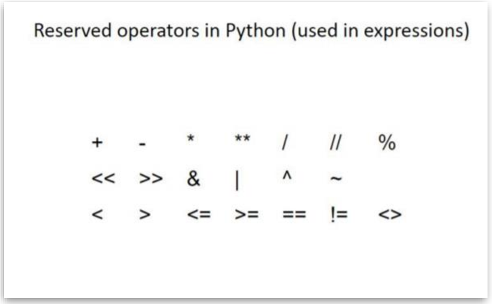
Python Reserved Words (Keywords)

You cannot use these words in **a variable** name

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

Python Operators

To be used in forming expressions.



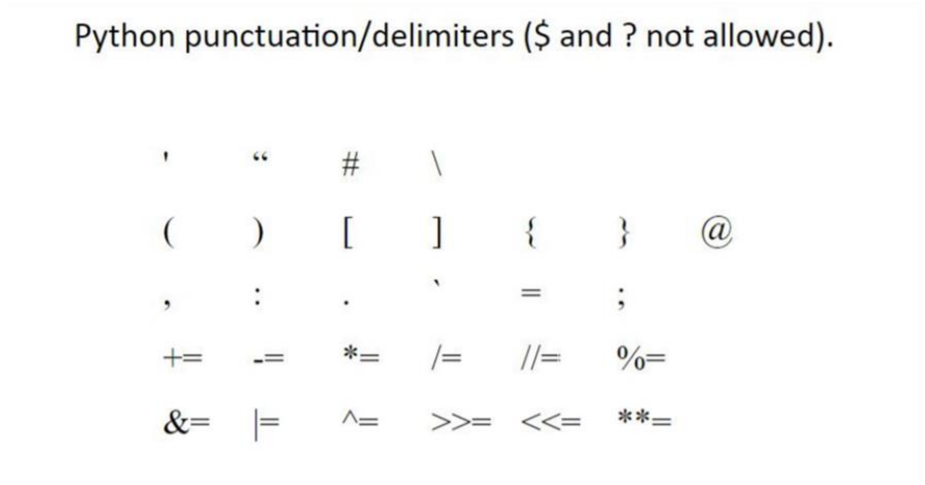
Reserved operators in Python (used in expressions)

+	-	*	**	/	//	%
<<	>>	&	|	^	~	
<	>	<=	>=	==	!=	<>

```
a + b # Penjumlahan

a - b # Pengurangan

a / b # Pembagian

a // b # Pembagian (Pembulatan ke bawah)

a * b # Perkalian

a ** b # Perpangkatan (A pangkat B)

a % b # Modulo
```

```
== # Sama dengan, return true/false

!= # Tidak sama dengan, return true/false

<> # Tidak sama dengan, return true/false

> # Lebih dari, return true/false

< # Kurang dari, return true/false

>= # Lebih dari sama dengan, return true/false

<= # Kurang dari sama dengan
```

Python Punctuators
(Tanda Baca)



Python punctuation/delimiters ($ and ? not allowed).

'	“	#	\			
(	)	[	]	{	}	@
,	:	.	`	=	;	
+=	-=	*=	/=	//=	%=	
&=	|=	^=	>>=	<<=	**=	

Literals

Literal is a programming notation for a **fixed value**.

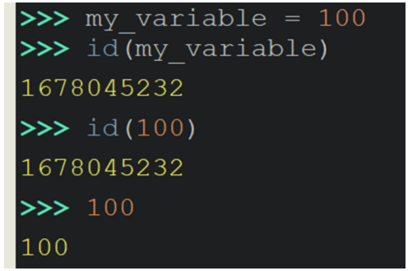
- ➔ For example, 123 is a fixed value, an integer.
- ➔ It would be weird if the symbol 123's value could change to be 3.14

Objects

In Python, every **“thing” in the system** is considered to be an object.

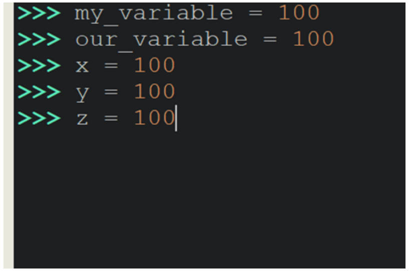
An object in Python has:

An identity (ID number)



```
>>> my_variable = 100
>>> id(my_variable)
1678045232
>>> id(100)
1678045232
>>> 100
100
```

Name(s)



```
>>> my_variable = 100
>>> our_variable = 100
>>> x = 100
>>> y = 100
>>> z = 100
```

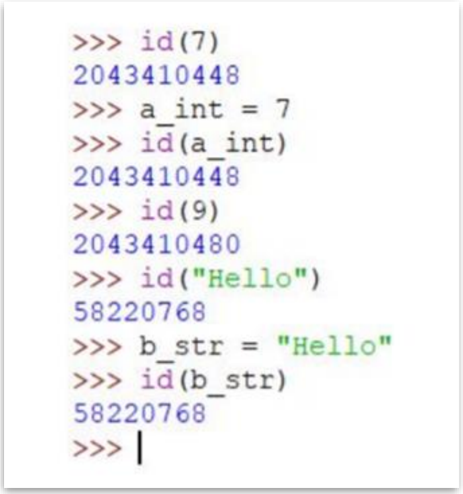
Some attributes

We will learn this later ^^

Each Object Has an ID

Each time an object is created, it **receives an ID number**.

- ➔ Used by Python to distinguish one object from another.
- ➔ Hard to remember, so we use name (such as variable name) to reference them



```
>>> id(7)
2043410448
>>> a_int = 7
>>> id(a_int)
2043410448
>>> id(9)
2043410480
>>> id("Hello")
58220768
>>> b_str = "Hello"
>>> id(b_str)
58220768
>>> |
```

Each Object Can Have Names (Variables)

A **variable** is a **name** we designate **to represent an object** (number, data structure, function, etc.) in our program

- ➔ We use names to make our program more readable, so that the object is easily understood in the program
- ➔ An object can have a name or even **multiple names**.
- ➔ Not part of object's ID.
- ➔ Used by us, programmers, to make code more readable.
- ➔ Python uses namespace to associate a name (such as variable name) with an object. Multiple names may be associated with the same object.

Object and Variable

Object is simply something to manipulate in a program

Variable is a name we designate to represent an object

Jar
A space in the memory

Nastar
The object

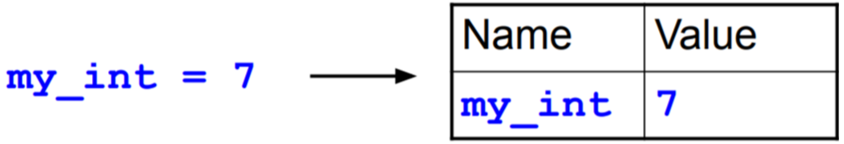
Label
Variable

Image credits to: IKEA, Detik Food, https://dribbble.com/shots/11944605-Label-Kue-Nastar-P1

Variable Creations

Python maintains a list of pairs for every variable:

variable's value	**variable's name**



```
my_int = 7
```

Name	Value
my_int	7

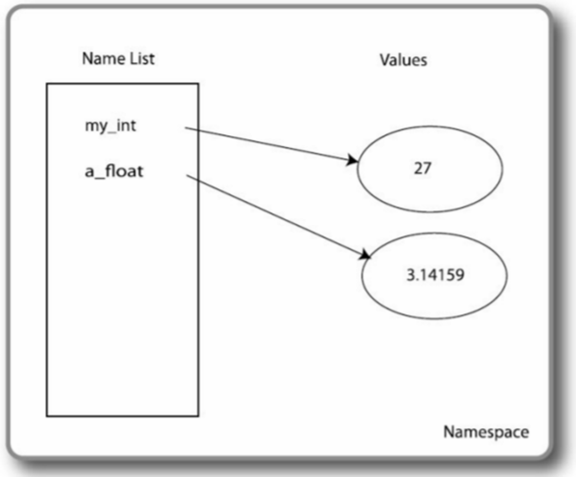
- ➔ A variable is created when a value is assigned the first time. It associates a name and a value
- ➔ Subsequent assignments update the associated value.
- ➔ We say “the name references a value”

Namespace

A namespace is the table that contains **the association of a name with a value**

We will see more about namespaces as we get further into Python,
but it is an essential part of the language

```
my_int=27
a_float=3.14159
```



A Simple Code: “Halo, `<Your_Name>`!” (1)

```
name = input("Name: ")
print("Hello, " + name + "!")
```

A Simple Code: “Halo, `<Your_Name>`!” (2)

```
name = input("Name: ")
print("Hello, " + name + "!")
```

```
Name: Hinata Shouyou
Hello, Hinata Shouyou!
```

A Simple Code: “Halo, `<Your_Name>`!” (3)

```
name = input("Name: ")
print("Hello, " + name + "!")
```

```
Name: Hinata Shouyou
Hello, Hinata Shouyou!
```

- ➔ `name` adalah suatu variable
- ➔ `input()` adalah suatu fungsi, yang:

  (1) mencetak "Name: " pada command line
  (2) menunggu masukan nama yang akan kita ketik (diakhiri Enter)
  (3) mengembalikan nama yang kita ketik sebagai suatu string

A Simple Code: "Halo, `<Your_Name>`!" (4)

```
name = input("Name: ")
print("Hello, " + name + "!")
```

```
Name: Hinata Shouyou
Hello, Hinata Shouyou!
```

- ➔ Tanda = menyatakan assignment:
  Nilai sebelah kanan diasosiasikan dengan variabel sebelah kiri
- ➔ Dengan kata lain, nama yang kita input akan di-assign ke variabel name
- ➔ Ingat, = **bukan math equality**

A Simple Code: “Halo, `<Your_Name>`!” (5)

```
name = input("Name: ")
print("Hello, " + name + "!")
```

```
Name: Hinata Shouyou
Hello, Hinata Shouyou!
```

- ➔ `print()` adalah suatu fungsi yang **mencetak pada command line**
- ➔ Dalam hal ini, yang dicetak adalah hasil "penambahan" antara: `"Hello, " + name + "!"`
- ➔ Penambahan yang terjadi di sini adalah: **string concatenation**
- ➔ Ingat, yang dicetak akan bervariasi bergantung pada input nama yang kita berikan! ^^

= is ASSIGNMENT

In many computer languages, = means assignment.

```
my_int = my_int + 7
left_hand_side = right_hand_side
```

What assignment means is:
1. evaluate the right-hand-side of the = sign
2. take the resulting value and associate it with the name on the left-hand-side

Examples

Example 1

```
my_var = 2 + 3 * 5
```

- ➔ evaluate expression (2+3*5): 17
- ➔ change the value of `my_var` to reference 17

Example 2

```
my_int = 2
my_int = my_int + 3
```

- ➔ evaluate expression (`my_int` + 3): 5
- ➔ change the value of `my_int` to reference 5

What’s happening in namespace
(Behind the scene)

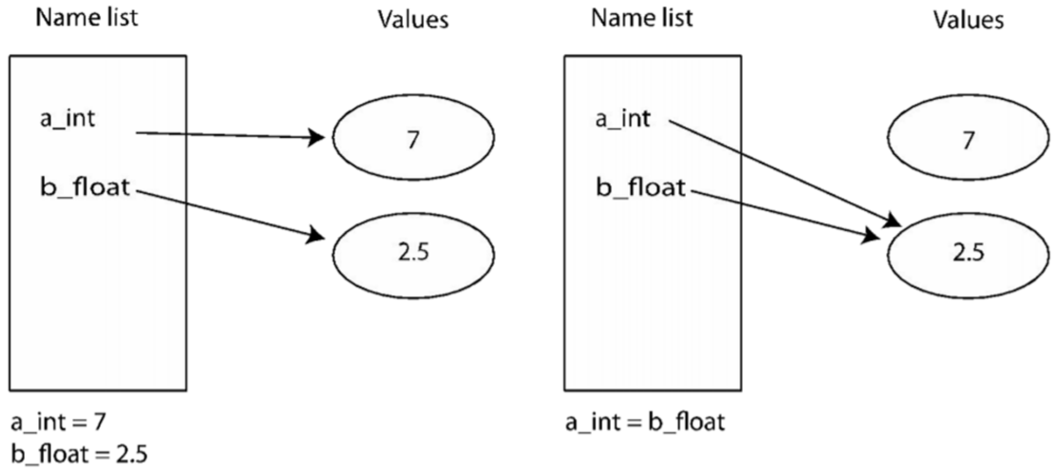
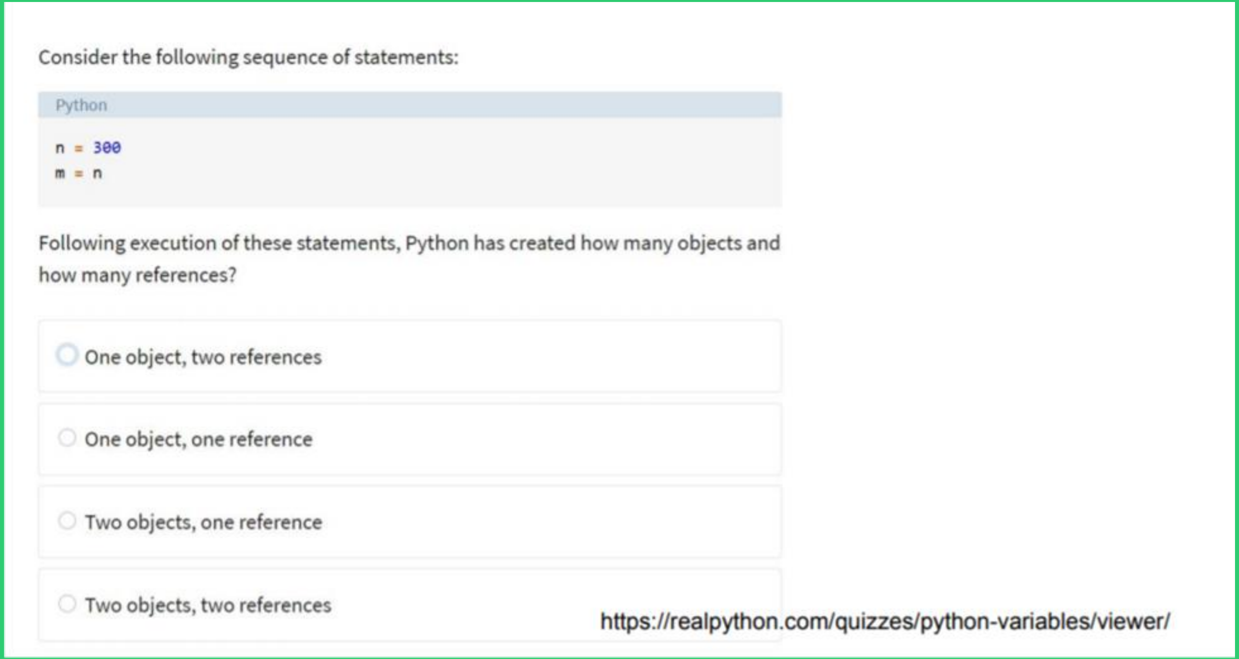


FIGURE 1.2 Namespace before and after the final assignment.

Triggering Question 1



Consider the following sequence of statements:

Python

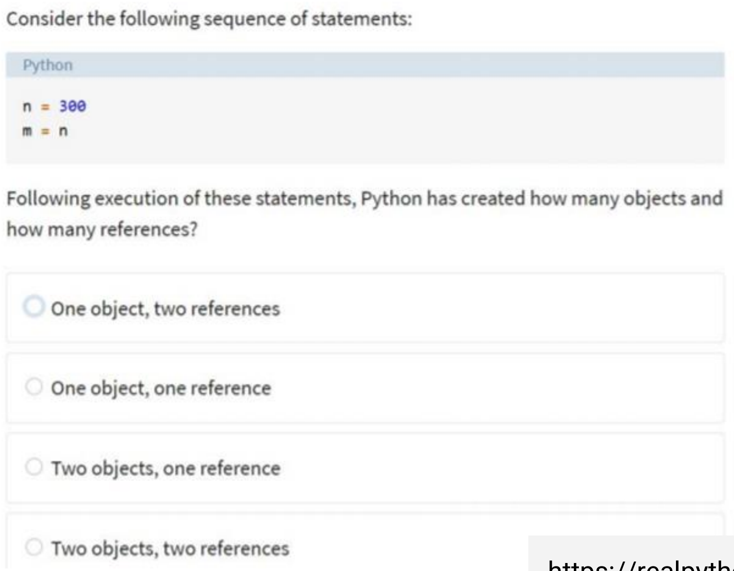
```
n = 300
m = n
```

Following execution of these statements, Python has created how many objects and how many references?

- One object, two references
- One object, one reference
- Two objects, one reference
- Two objects, two references

https://realpython.com/quizzes/python-variables/viewer/

Objects and References



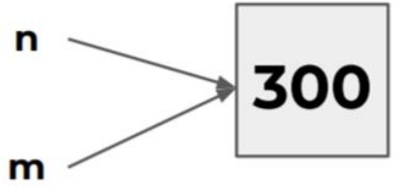
Consider the following sequence of statements:

Python

```
n = 300
m = n
```

Following execution of these statements, Python has created how many objects and how many references?

- One object, two references
- One object, one reference
- Two objects, one reference
- Two objects, two references



$$\begin{tikzcd} n \arrow[dr] & \\ & \boxed{300} \\ m \arrow[ur] & \end{tikzcd}$$

https://realpython.com/quizzes/python-variables/viewer/

A Simple Calculator: Area of a Circle $A = \pi r^2$

```
radius = 10
print(3.14 * radius * radius)
```

Atau...

```
radius = 10
print(3.14 * radius **2)
```

```
314.0
```

➔ Kita juga dapat menggunakan Python sebagai calculator, dengan bermacam pilihan operator: +, -, *, /, %, **, //

Triggering Question 2

**Print the Volume of a cube,
given that the value of the side is
inputted by the user**

Variables Do Not Have Type

Python **does not require you to pre-define what type** can be associated with a variable

- ➔ What type a variable holds can change
- ➔ Nonetheless, knowing the type can be important for using the correct operation on a variable. Thus proper naming is important

Objects (Values) **Have Types**

➔ Use `type()`

➔ Type of variables is just the type of the object/value they are associated with, and they can change. This is called **dynamic typing**.

```
#How To Check Data Type

type(variable_name)

#Output: the type of variable
```

Python data types:

Number: integer, float, complex	Dictionary
String	Set
Boolean	Tuples
List	

Dynamic Typing in Python

You can assign variable with **multiple data type**

```
#Dynamic typing

a = 100

a = 'DDP 1'

a = 100.129

a = True

print(a)
```

Importance of knowing types

Knowing the type is important in Python since it defines two things:

The internal structure of the object (**What it contains**)

➔ No decimal point in `int` objects, no letters in a `float` and `int` object

the **kinds of operations** you can perform on the object

➔ We can divide `int` objects, but cannot do division on `str` objects
➔ `'abc'.capitalize()` works on `str` object such as `'abc'`, but not on `int` objects

Boolean

- ➔ Only consists of two values. **True** or **False**
- ➔ Usually used to compare data

```
a = True
b = False

a or b
a and b
not b
not a or b
```

Numbers in Python

- ➔ **int**

```
a = 10
```

- ➔ **float**

```
a = 10.12938
```

- ➔ **complex**

```
a = 10+0j
```

Integer can be converted to float and vice versa.
Both integer and float can be converted to complex number but complex number can’t be converted

Strings

String initiation

```
name = 'Midoriya Izuku'
```

String manipulation

```
print(name[2:])
print(name[:5])
print(name[4])
print(name[2:8])
print(name[2:20])
```

```
print(len(name))
print(name.capitalize())
print(name.lower())
```

We will learn more about this later ^^

M	**i**	**d**	**o**	**r**	**i**	**y**	**a**		**I**	**z**	**u**	**k**	**u**
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Type Conversion (1)

A character `'1'` is not an integer `1`.
We'll see more on this later, but take my word for it.

- ➔ You need to convert the value returned by the `input` command (characters) into an integer
- ➔ `int("123")` yields the integer `123`

Type Conversion (2)

- ➔ `int(some_var)` returns an integer
- ➔ `float(some_var)` returns a float
- ➔ `str(some_var)` returns a string

should check out what works:

- ➔ `int(2.1)` → 2
- ➔ `int('2')` → 2
- ➔ but `int('2.1')` **fails**

- ➔ `float(2)` → 2.0
- ➔ `float('2.0')` →2.0
- ➔ `float('2')` →2.0
- ➔ `float(2.0)` →2.0

- ➔ `str(2)` →'2'
- ➔ `str(2.0)` →'2.0'
- ➔ `str('a')` →'a'

Naming Convention (1)

➔ must **begin** with a letter or underscore _ — `Ab_123` is OK, but `123_ABC` is not.

➔ may contain **letters**, **digits**, and **underscores** — `this_is_an_identifier_123`

➔ may be of **any length**

➔ **upper** and **lower case** letters are different — `Length_Of_Rope` is not `length_of_rope`

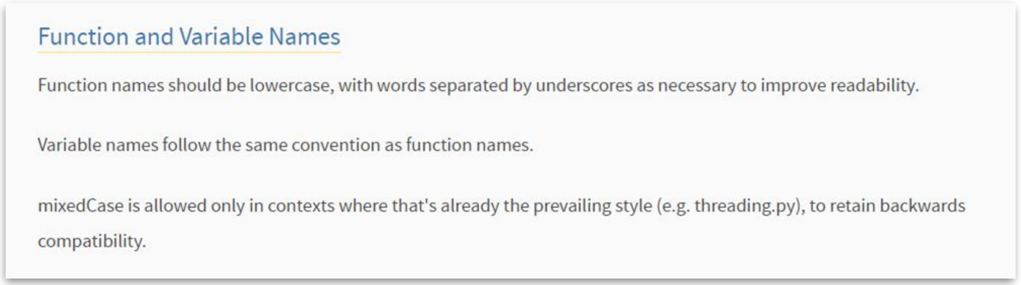
➔ names starting with _(underline) have special meaning.
Be careful!

Naming Convention (2)

Fully described by PEP8 or Google Style Guide for Python https://www.python.org/dev/peps/pep-0008/#id45

the standard way for most things named in python is **lower with underline**, lower case with separate words joined by an underline:

- ➔ `this_is_a_var`
- ➔ `my_list`
- ➔ `square_root_function`



Function and Variable Names

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

Variable names follow the same convention as function names.

mixedCase is allowed only in contexts where that's already the prevailing style (e.g. threading.py), to retain backwards compatibility.

Review Questions

What is object in Python?

What is variable in Python?

Mention and explain some Python data types.

Explain what dynamic typing is in Python?

Analyze the following code.
How many objects and references will be created if we run the code?

```
x = 200
y = 5000
z = y
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Q&A Session