

# Intro to Computer Science

Summer 2020

August 4th to August 20th, online

## Variables and Expressions

# Today's questions

How do computers conduct tasks we ask for?

How do computers store information (data) using code?

Once we store that information, how do we use it?

# Today's topics

1. Welcome to Python  
Input, output, process
2. Variables  
Assignment and retrieval  
Types
3. Using variables  
In expressions

# Welcome to Python

# Welcome to Python

Guido van Rossum  
(Creator of Python)



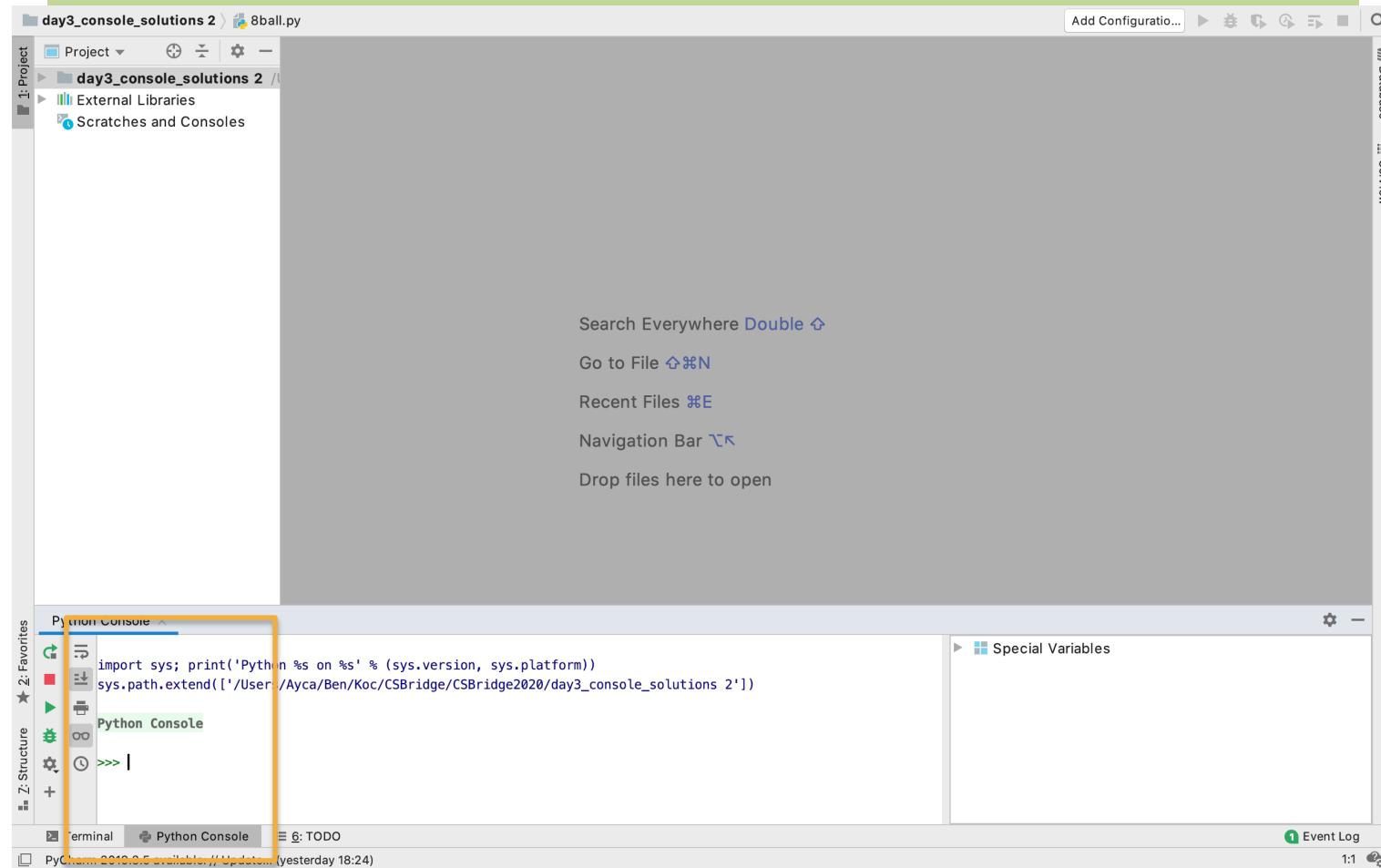
# Using Python

- **Python must be installed and configured prior to use**
  - One of the items installed is the Python interpreter
- **Python interpreter can be used in two modes:**
  - Interactive mode: enter statements on keyboard
  - Script mode: save statements in Python script

# Interactive Mode in Python

- **When you start Python in interactive mode, you will see a prompt**
  - Indicates the interpreter is waiting for a Python statement to be typed
  - Prompt reappears after previous statement is executed
  - Error message displayed If you incorrectly type a statement
- **Good way to learn new parts of Python**

# Interactive Mode in Python



# Writing and Running in Script Mode

- **Statements entered in interactive mode are not saved as a program**
- **To have a program use script mode**
  - Save a set of Python statements in a file
  - The filename should have the .py extension
  - To run the file, or script, type  
`python filename`  
at the operating system command line

# Writing and Running in Script Mode

The screenshot shows the PyCharm IDE interface. The top navigation bar displays the project path: Lecture4-Python [/Users/Ayca/Ben/Koc/CSBridge/CSBridge2020/Lecture4-Python] - /Users/Ayca/Ben/Koc/CSBridge/CSBridge2020/Day3-Vari... and the file Day3-Variables > helloworld.py. The main window has tabs for helloworld.py, constants.py, roldice.py, helloworldStarter.py, and intro. The left sidebar shows the Project tree with Day3-Variables containing files like add2numbers.py, add2numbersStarter.py, constants.py, constantsStarter.py, helloworld.py (selected), helloworldStarter.py, roldice.py, and roldiceStarter.py. External Libraries and Scratches and Consoles are also listed. The SciView and Database tabs are visible on the right.

File: helloworld.py

This is our first python program. It is customary to have a programmer's first program write "hello world" (inspired by the first program in Brian Kernighan and Dennis Ritchie's classic book, 'The C Programming Language.')

```
1     """
2     File: helloworld.py
3
4     This is our first python program. It is customary to
5     have a programmer's first program write "hello world"
6     (inspired by the first program in Brian Kernighan and
7     Dennis Ritchie's classic book, 'The C Programming Language.')
8     """
9
10
11    def main():
12        print('hello, world!')
13
14
15    # This provided line is required at the end of a Python file
16    # to call the main() function.
17    if __name__ == '__main__':
18        main()
```

Terminal: Local +

Unable to find any JVMs matching version "1.5".

The default interactive shell is now zsh.  
To update your account to use zsh, please run `chsh -s /bin/zsh`.  
For more details, please visit <https://support.apple.com/kb/HT208050>.

(base) Aycas-MacBook-Pro:Day3-Variables ayildirim\$ python3 helloworld.py

How do computer program typically consists of?

# Input, Pro



# **Input, Processing and Output**

- **Typically, computer performs three-step process**
  - Receive input
    - Input: any data that the program receives while it is running
  - Perform some process on the input
    - Example: mathematical calculation
  - Produce output

# How do computers output?

# print function

```
print("This program adds two numbers.")
```

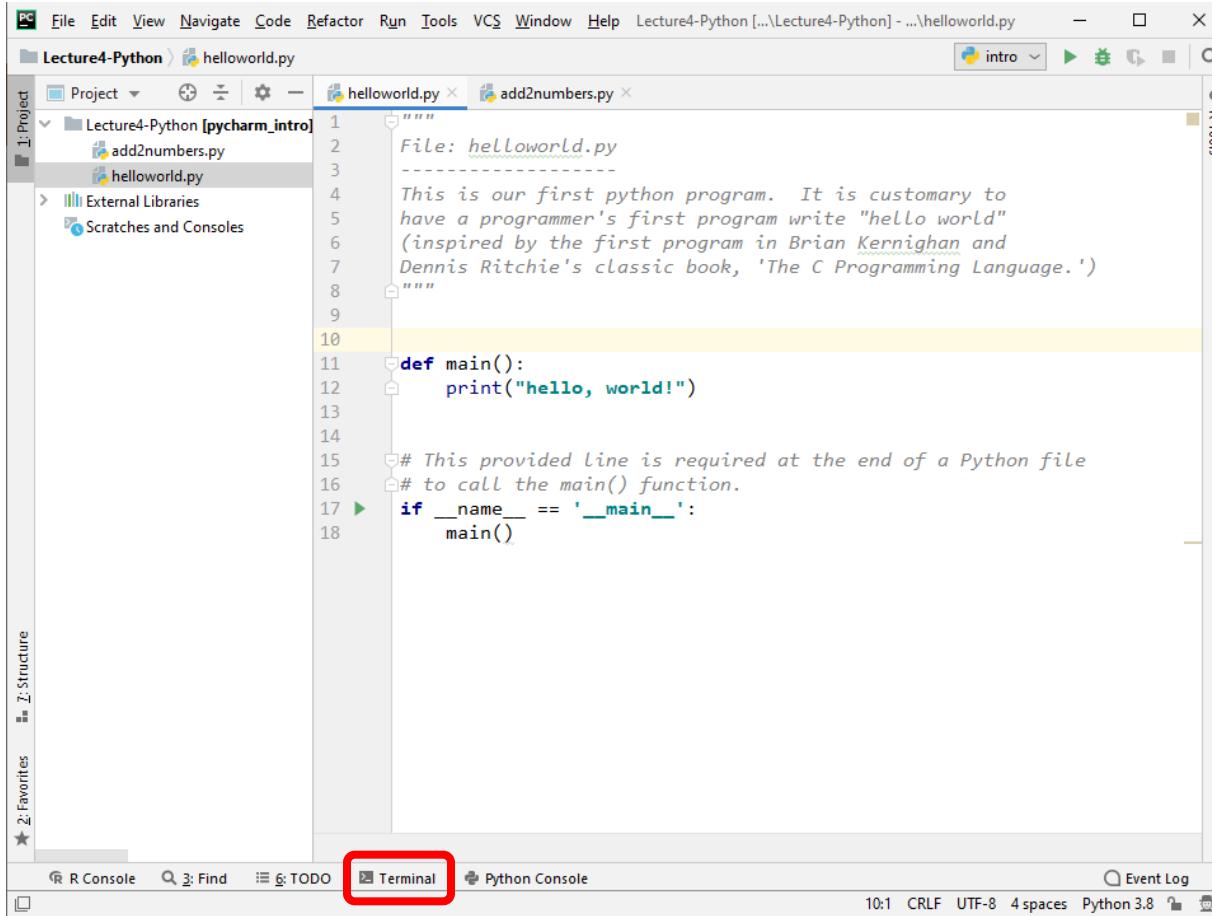
- **print** command prints text to the terminal
- Text printed is between double quotes ("text")
  - Can also be between single quotes ('text')
  - Choice of quotes depends on text you are printing
    - Double quotes when text contains single quotes  
`print("no, you didn't") → no, you didn't`
    - Single quotes when text contains double quotes  
`print('say "hi" Karel') → say "hi" Karel`

# Our first program



# Our First Python Program

# Our First Python Program



The screenshot shows the PyCharm IDE interface with the following details:

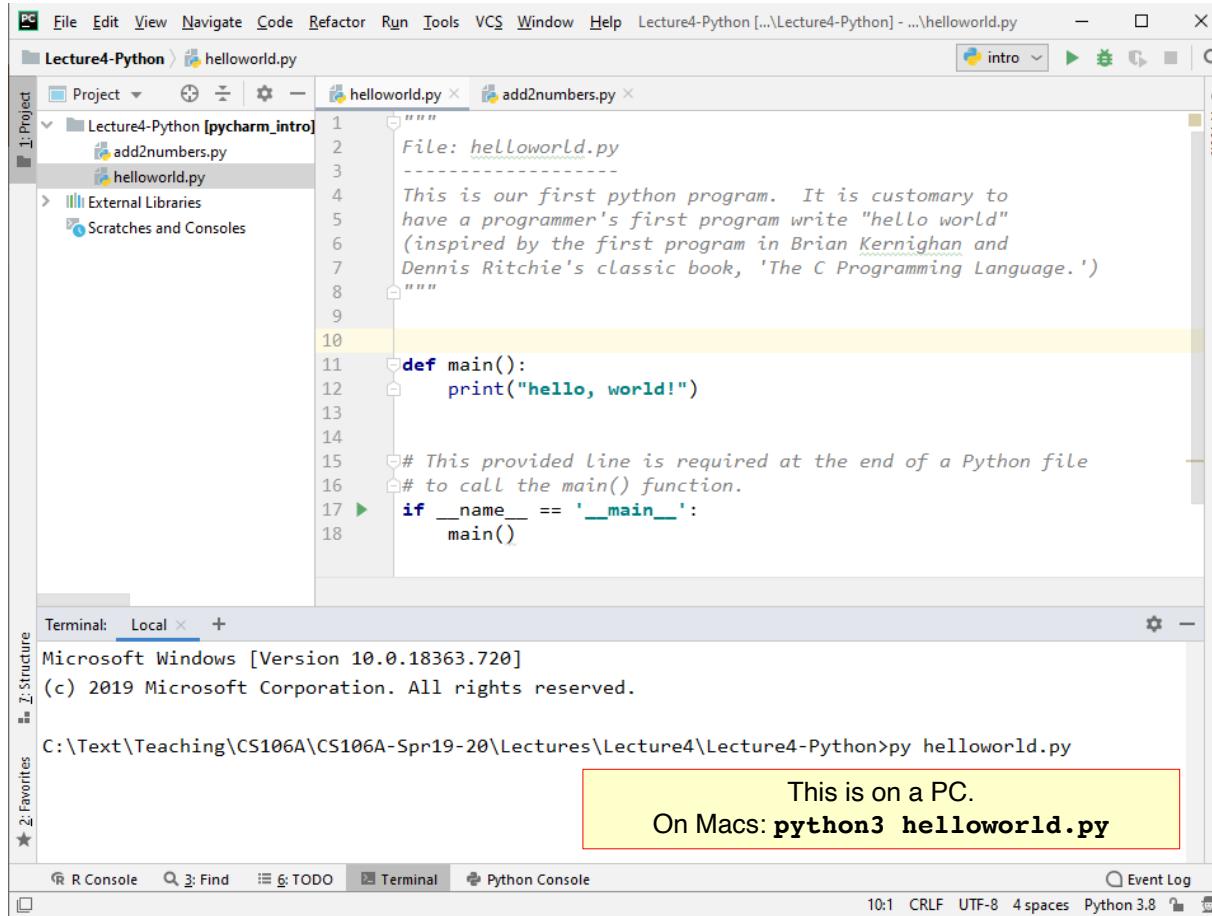
- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Title Bar:** Lecture4-Python [pycharm\_intro] - ...\\helloworld.py
- Toolbar:** intro, Run, Tools, Find, Event Log.
- Project Explorer:** Shows the project structure with files: Lecture4-Python [pycharm\_intro] (add2numbers.py, helloworld.py), External Libraries, and Scratches and Consoles.
- Code Editor:** Displays the content of helloworld.py:

```
"""
File: helloworld.py
-----
This is our first python program. It is customary to
have a programmer's first program write "hello world"
(inspired by the first program in Brian Kernighan and
Dennis Ritchie's classic book, 'The C Programming Language.')
"""

def main():
    print("hello, world!")

# This provided line is required at the end of a Python file
# to call the main() function.
if __name__ == '__main__':
    main()
```
- Bottom Navigation Bar:** R Console, Find, TODO, Terminal (highlighted with a red box), Python Console, Event Log.
- Status Bar:** 10:1 CRLF UTF-8 4 spaces Python 3.8

# Our First Python Program



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar indicates "Lecture4-Python [...] Lecture4-Python - ...\\helloworld.py". The left sidebar shows a Project tree with "Lecture4-Python [pycharm\_intro]" containing "add2numbers.py" and "helloworld.py", along with External Libraries and Scratches and Consoles. The main editor window displays the "helloworld.py" file:

```
"""  
File: helloworld.py  
-----  
This is our first python program. It is customary to  
have a programmer's first program write "hello world"  
(inspired by the first program in Brian Kernighan and  
Dennis Ritchie's classic book, 'The C Programming Language.')  
"""  
  
def main():  
    print("hello, world!")  
  
# This provided line is required at the end of a Python file  
# to call the main() function.  
if __name__ == '__main__':  
    main()  
  
Terminal: Local +  
Microsoft Windows [Version 10.0.18363.720]  
(c) 2019 Microsoft Corporation. All rights reserved.  
  
C:\Text\Teaching\CS106A\CS106A-Spr19-20\Lectures\Lecture4\Lecture4-Python>py helloworld.py
```

A yellow callout box in the bottom right corner contains the text: "This is on a PC.  
On Macs: **python3 helloworld.py**".

# Our First Python Program

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Title Bar:** Lecture4-Python [...]\Lecture4-Python - ...\\helloworld.py
- Toolbar:** intro, Run, Stop, Refresh, Find, Event Log.
- Project Explorer:** Shows the project structure under 'Lecture4-Python [pycharm\_intro]': add2numbers.py and helloworld.py.
- Code Editor:** Displays the content of helloworld.py:

```
"""  
File: helloworld.py  
-----  
This is our first python program. It is customary to  
have a programmer's first program write "hello world"  
(inspired by the first program in Brian Kernighan and  
Dennis Ritchie's classic book, 'The C Programming Language.')  
"""  
def main():  
    print("hello, world!")  
  
# This provided line is required at the end of a Python file  
# to call the main() function.  
if __name__ == '__main__':  
    main()
```
- Terminal:** Local, Microsoft Windows [Version 10.0.18363.720], (c) 2019 Microsoft Corporation. All rights reserved.
- Output Terminal:** C:\Text\Teaching\CS106A\CS106A-Spr19-20\Lectures\Lecture4\Lecture4-Python>py helloworld.py, hello, world!
- Bottom Navigation:** R Console, Find, TODO, Terminal, Python Console, Event Log.
- Status Bar:** 10:1, CRLF, UTF-8, 4 spaces, Python 3.8.



# You're now all Python programmers!

hey\_that\_looks  
like\_what\_I  
taught\_them()

# How do computers get input (data)?

# How do computers store information (data)?

# Your computer has memory!

- Information is stored in your computer's memory (RAM)



# How do computers store information (data) in code?

How do computers store information (data) in code?

Variables!

# Variable

## Definition

### **variable**

A way for code to store information by associating a value with a name

# Variable

Think of them as labels  
for containers!

## Definition

**variable**

A way for code to store information by  
associating a value with a name

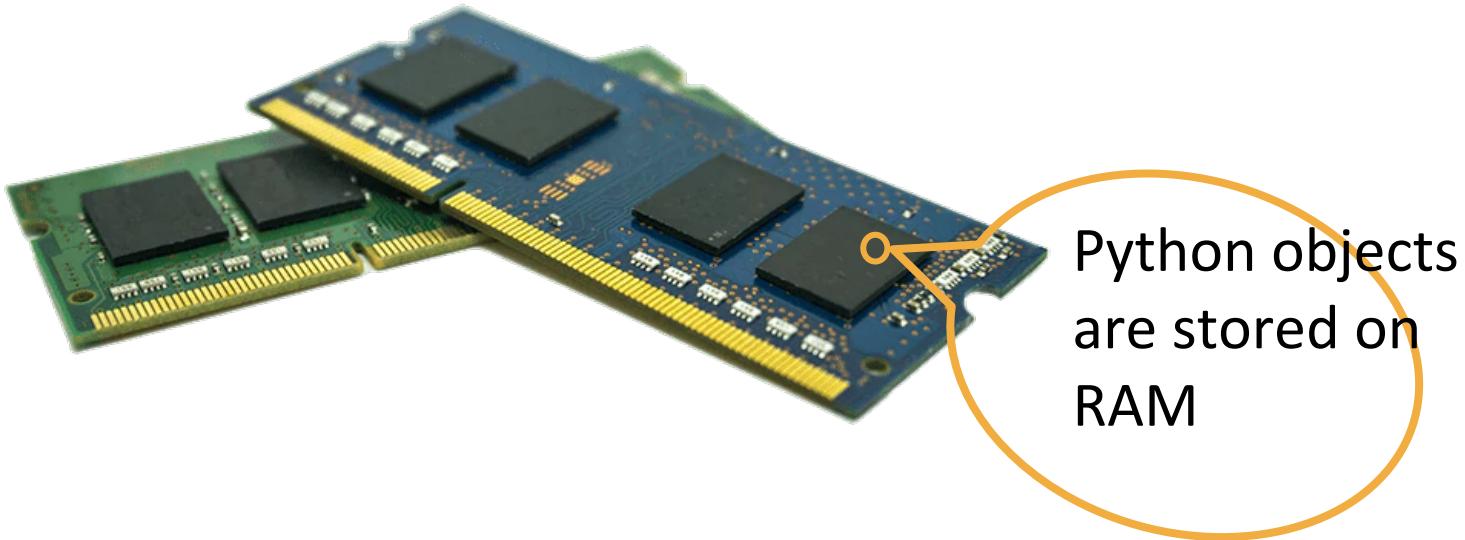
# Suitcase Analogy

- When you store information in Python, it becomes a Python **object**
  - Objects come in different sizes and types (more on types later)



# Suitcase Analogy

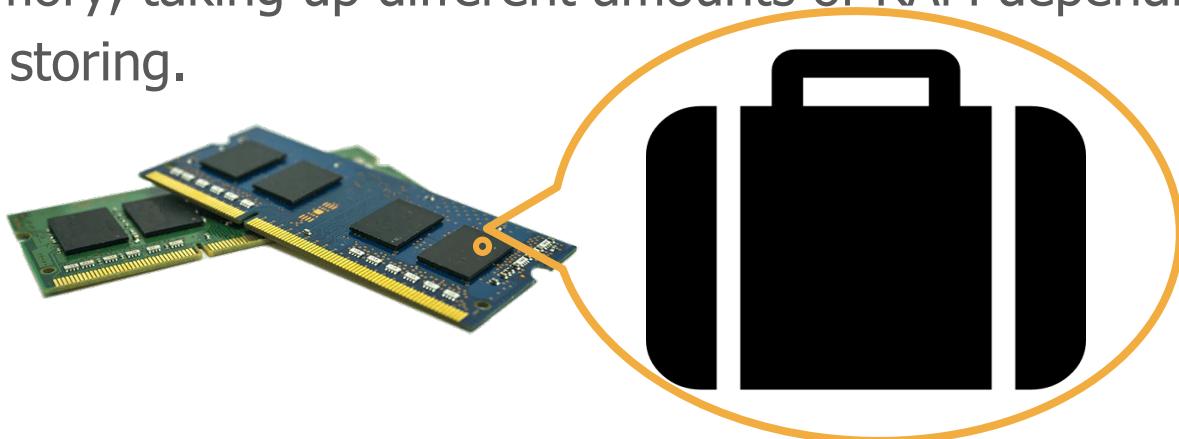
- When you store information in Python, it becomes a Python **object**
  - Objects come in different sizes and types (more on types later)



Python objects  
are stored on  
RAM

# Suitcase Analogy

- When you store information in Python, it becomes a Python **object**
  - Objects come in different sizes and types (more on types later)
- You can think about a Python object as a suitcase stored in your computer's memory, taking up different amounts of RAM depending on what you're storing.



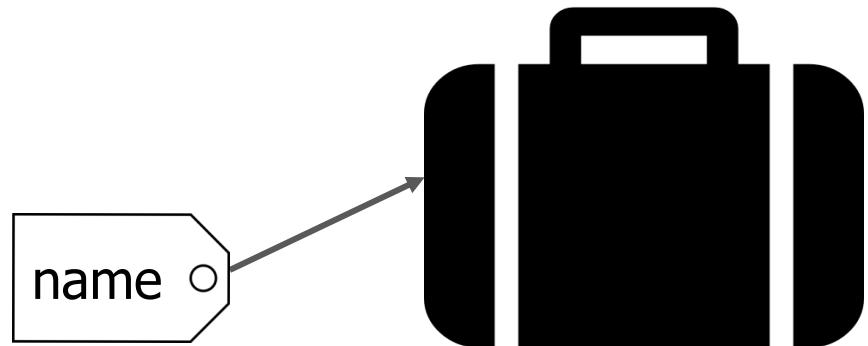
# Suitcase Analogy

- You can think about a Python object as a suitcase stored in your computer's memory.



# Suitcase Analogy

- A variable is a luggage tag for your suitcase that gives it a name!



# Variable

- **Variable**: name that represents a value stored in the computer memory
  - Used to access and manipulate data stored in memory
  - A variable references the value it represents
- **Assignment statement**: used to create a variable and make it reference data
  - General format is variable = expression
    - Example: age = 29
    - Assignment operator: the equal sign (=)

# Variable

- In assignment statement, variable receiving value must be on left side
- You can only use a variable if a value is assigned to it
  - my\_age = 18

# Variable Naming Rules

- **Rules for naming variables in Python:**
  - Variable name cannot be a Python key word
  - Variable name cannot contain spaces
  - First character must be a letter or an underscore
  - After first character may use letters, digits, or underscores
  - Variable names are case sensitive
- **Variable name should reflect its use**
  - $x = 10$  versus  $my\_grade = 10$

# A Variable Example

## An example

Suppose you're writing a program that keeps track of the flowers in your garden:



# A Variable Example

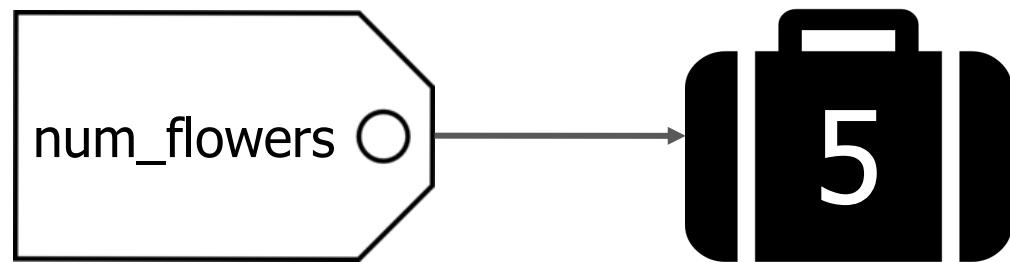
Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

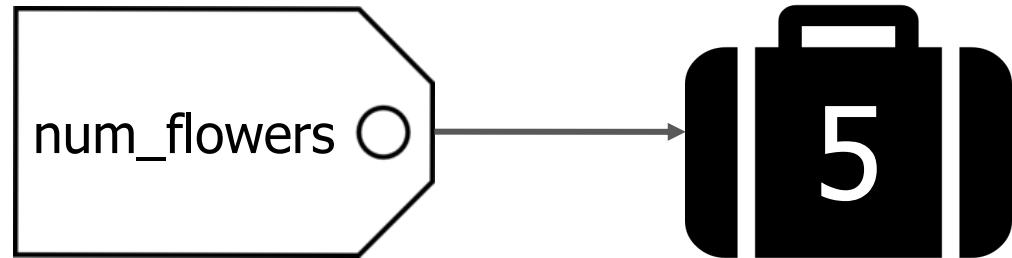
```
num_flowers = 5
```



# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```



**Definition**

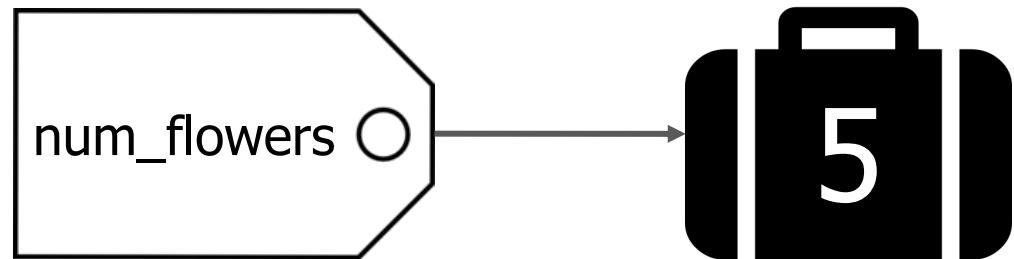
## **variable assignment**

The process of associating a name with a value (use the `=`)

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```



**Definition** i.e. attaching it to the bag

**variable assignment**

The process of associating a name with a value (use the `=`)

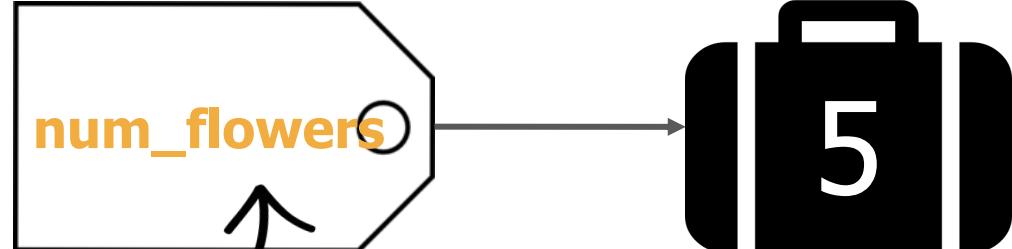
# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```



variable's  
name



# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```

num\_flowers

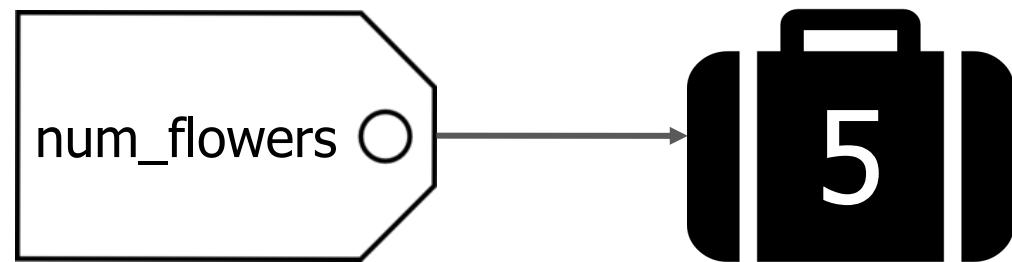
5

variable's  
value

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

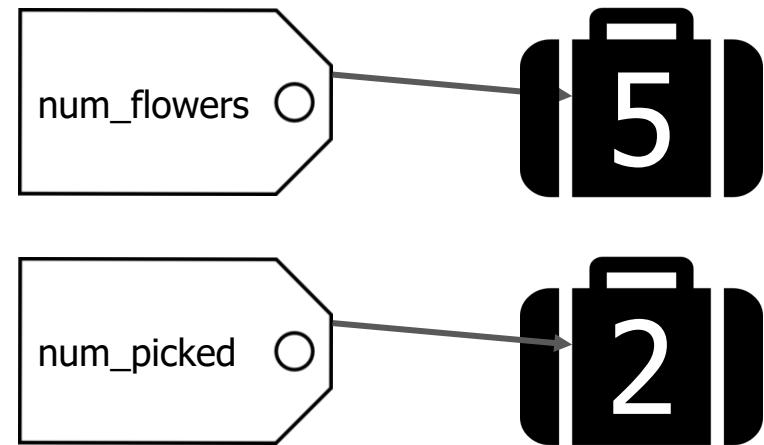
```
num_flowers = 5
```



# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2
```



# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```

```
num_picked = 2
```

```
num_flowers = num_flowers - num_picked
```

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```

```
num_picked = 2
```

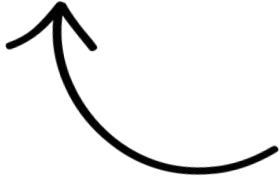
```
num_flowers = num_flowers - num_picked
```

**Think/Share:** Try to predict what happens here!

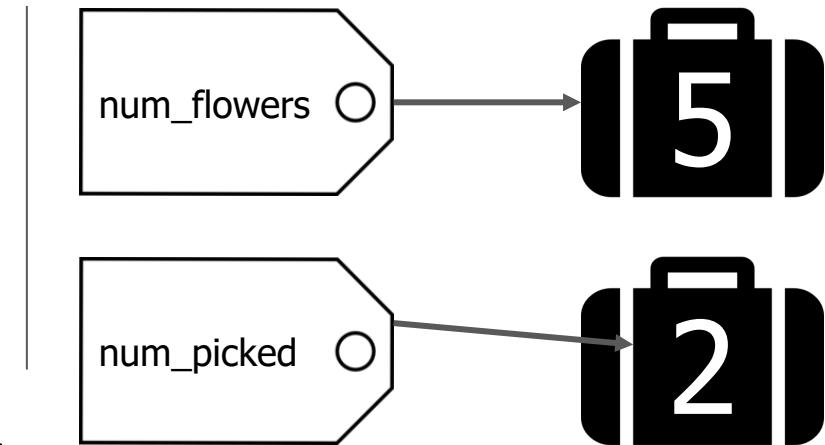
# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```



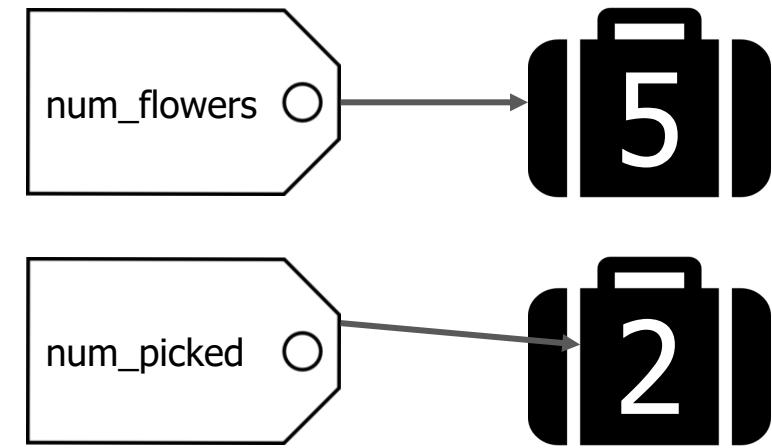
variable  
assignment!



# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```



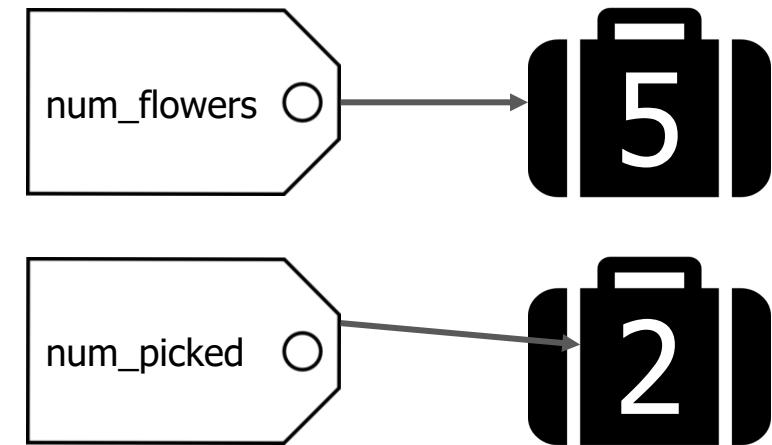
The right side of the equals sign **always** gets evaluated first.

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```

variable  
retrieval!

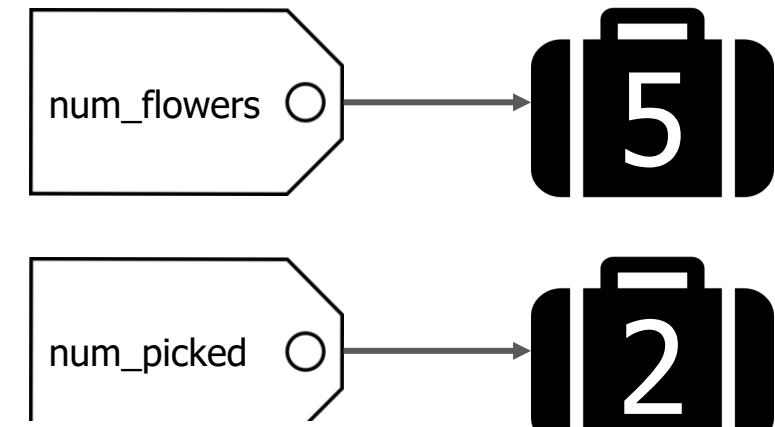


# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```

## Definition



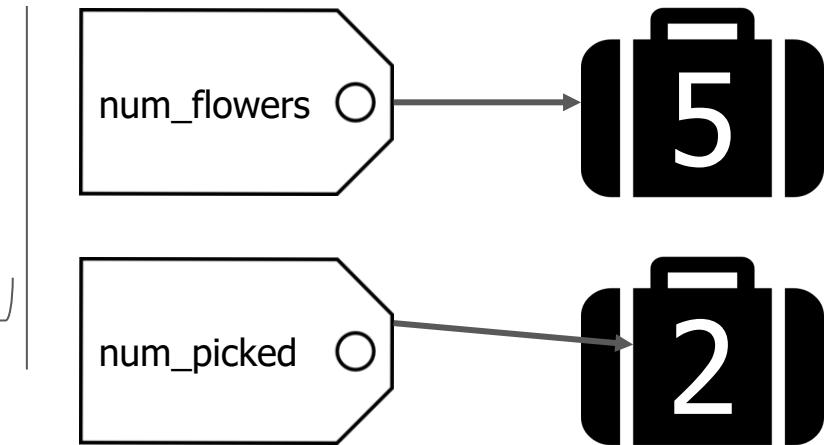
## variable retrieval

The process of getting the value associated with a name

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```

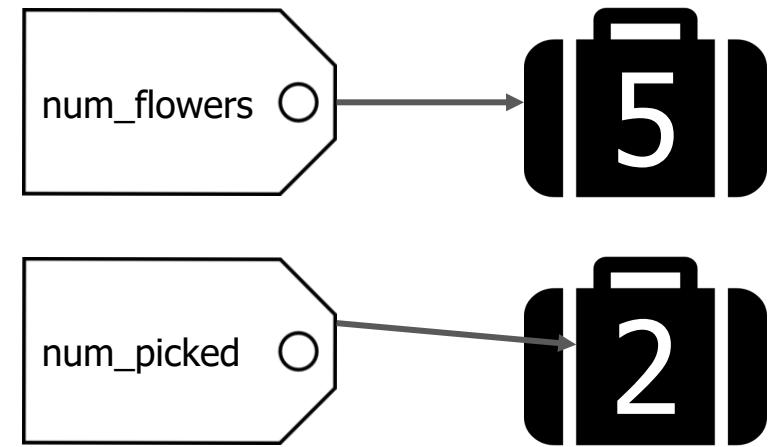


The right side of the equals sign **always** gets evaluated first.

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```



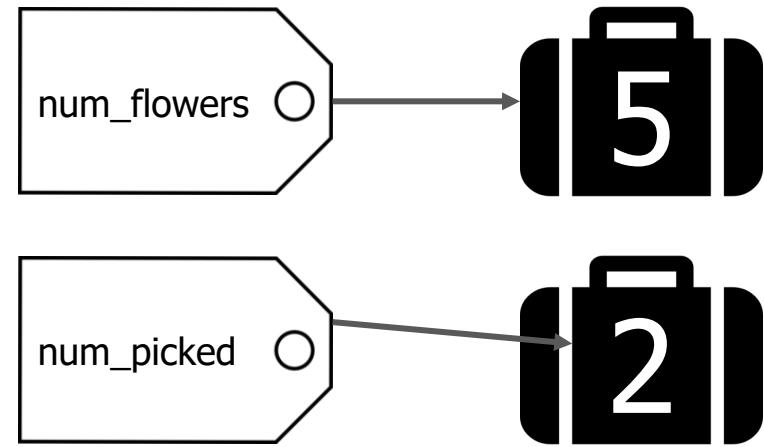
We get the values using variable retrieval (i.e. checking what suitcase is attached).

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```

The code shows three assignments. The first two are simple variable declarations. The third assignment is a subtraction operation where 'num\_flowers' is being set to the result of 'num\_flowers' minus 'num\_picked'. Arrows point from the variable names in the assignment to their corresponding values: 'num\_flowers' points to '5', and 'num\_picked' points to '2'.



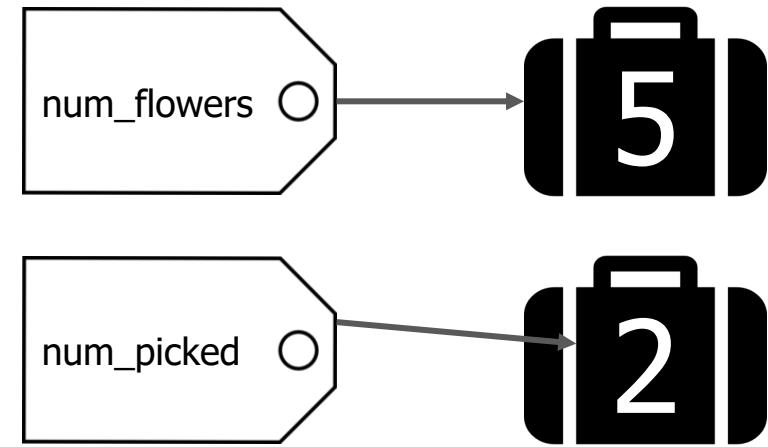
Then we can evaluate the right hand side of the assignment.

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers = num_flowers - num_picked
```

The diagram illustrates the state of variables in memory. On the left, three variables are defined: `num_flowers` is 5, `num_picked` is 2, and a third unnamed variable is 3. Arrows point from the values 5 and 2 to the respective variable names. A bracket groups the unnamed variable and the expression `num_flowers - num_picked`, with an arrow pointing to the value 3.



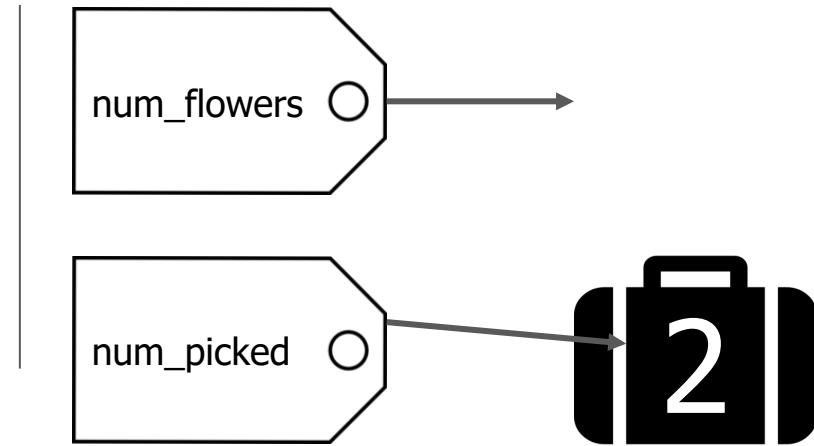
Then we can evaluate the right hand side of the assignment.

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers =
```

3



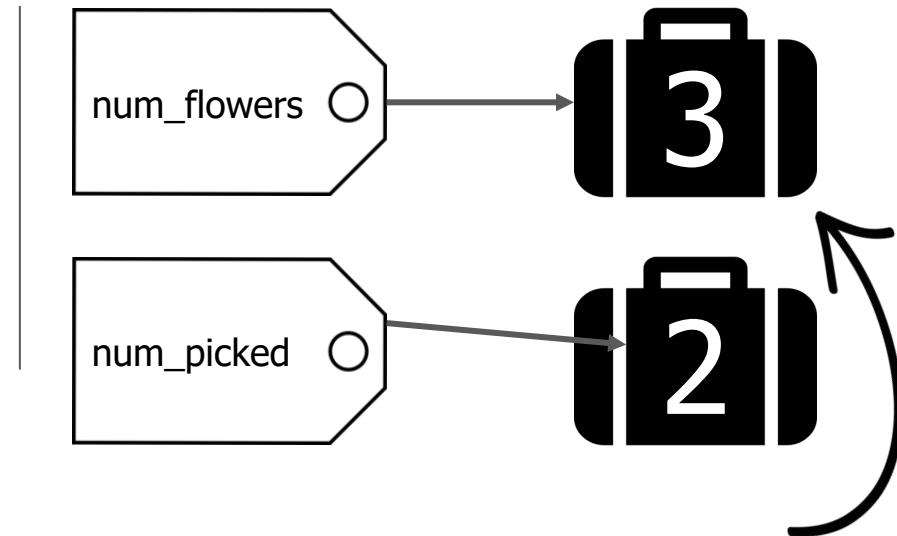
The right side of the equals sign **always** gets evaluated first.

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers =
```

3



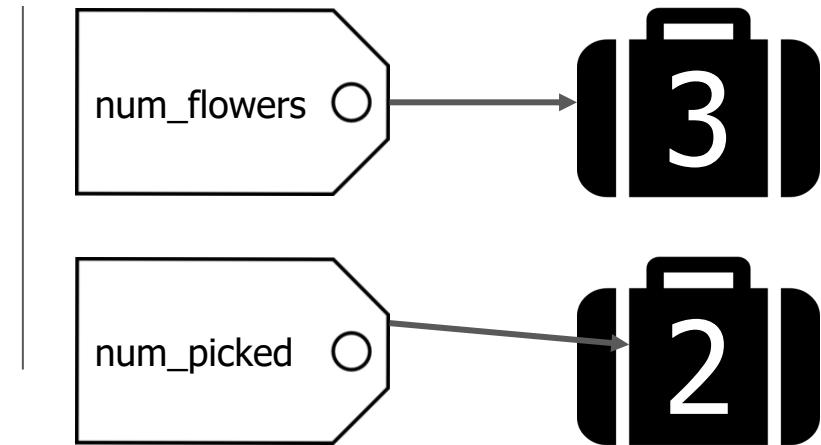
This is a new Python object!

# A Variable Example

Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5  
num_picked = 2  
num_flowers =
```

3



Python handles all the baggage for you when you use variables.

# How do computer get user input?

# input function

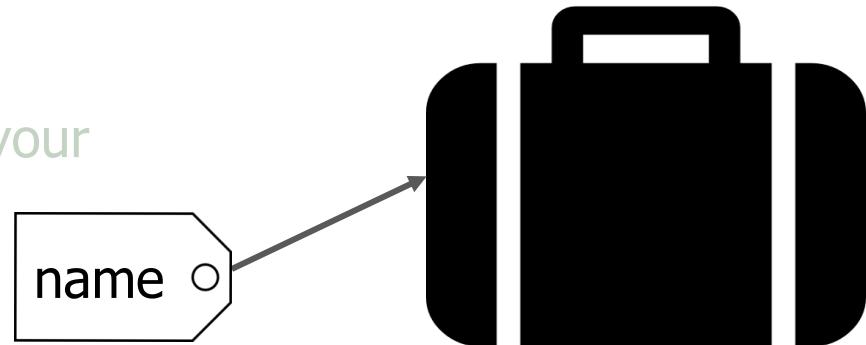
```
num1 = input("Enter first number: ")
```

- **input** command gets text input from the user
- Prints text specified in double/single quotes
  - Then waits for user input
  - Here, user input from **input** is put in a variable (**num1**)
  - The user input is considered text, even if user entered a number
- We'll talk more about **input** function later

# Data Types

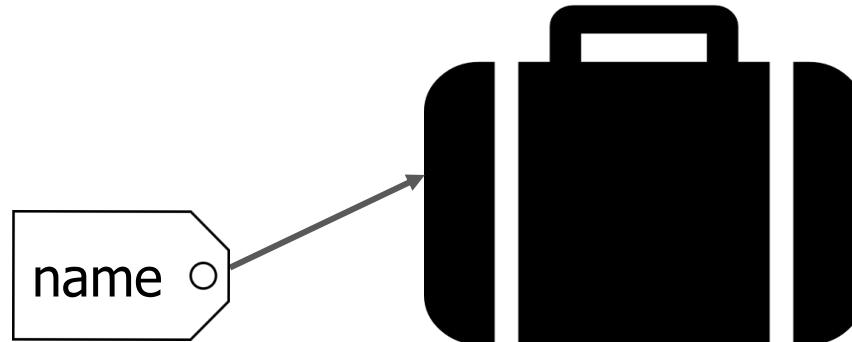
# The suitcase analogy

- When you store information in Python, it becomes a Python **object**
  - **Objects come in different sizes and types**
- You can think about a Python object as a suitcase stored in your computer's memory.
- A variable is a luggage tag for your suitcase that gives it a name!



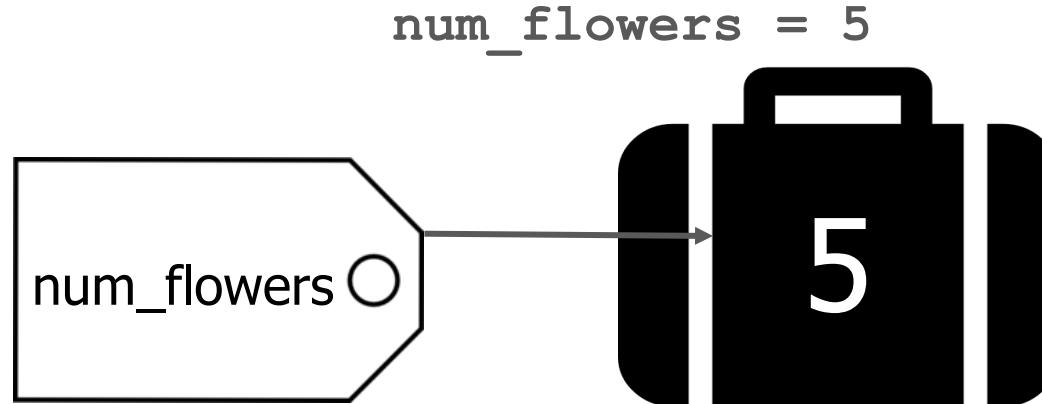
# The suitcase analogy

- When you store information in Python, it becomes a Python **object**
  - **Objects come in different sizes and types**



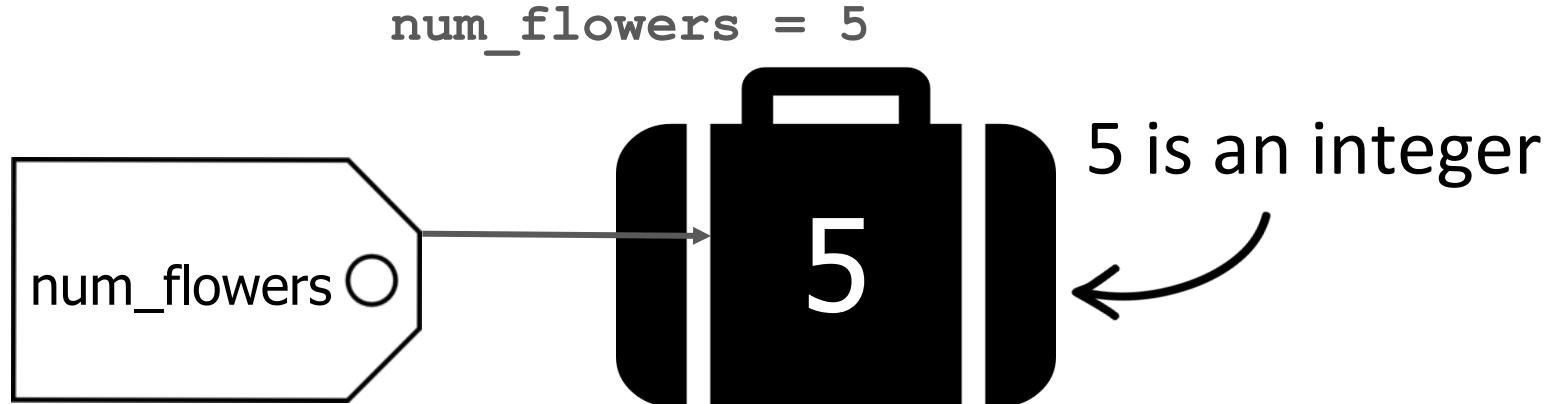
# The suitcase analogy

- When you store information in Python, it becomes a Python **object**
  - **Objects come in different sizes and types**



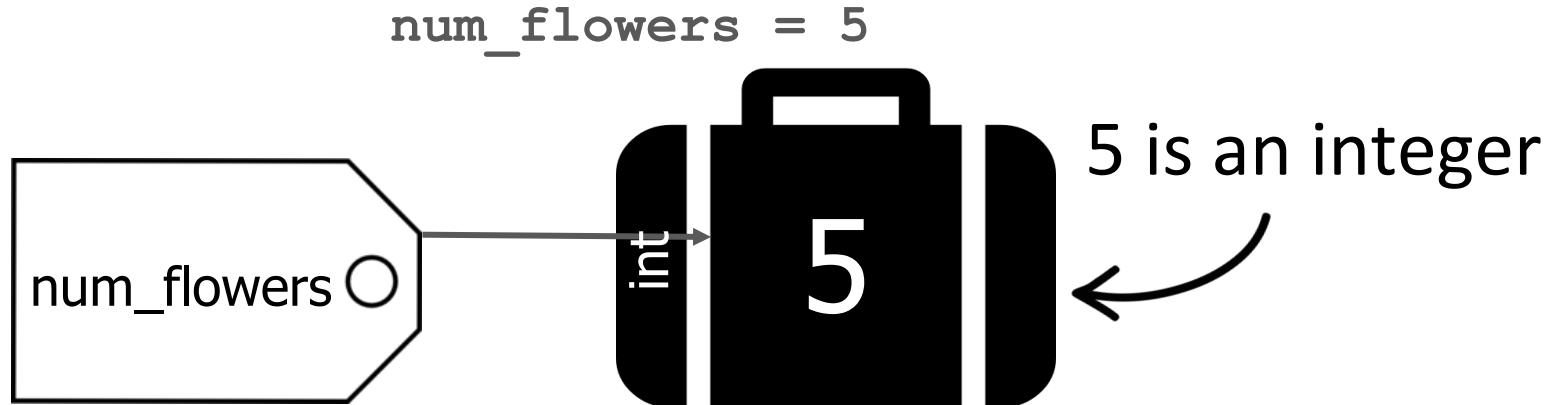
# The suitcase analogy

- When you store information in Python, it becomes a Python **object**
  - **Objects come in different sizes and types**



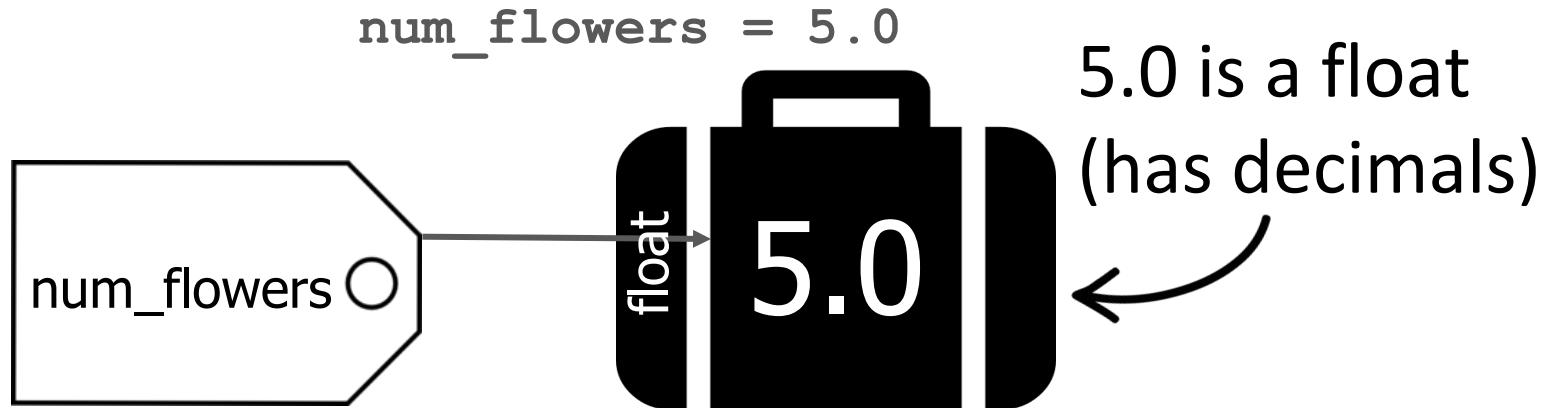
# The suitcase analogy

- When you store information in Python, it becomes a Python **object**
  - **Objects come in different sizes and types**



# The suitcase analogy

- When you store information in Python, it becomes a Python **object**
  - Objects come in different sizes and types**



# Types

All Python objects have a type!

- Python automatically figures out the type based on the value
  - Variables are “**dynamically-typed**”: you don’t specify the type of the Python object they point to

# Types

All Python objects have a type!

- Integers - numbers with no decimals

```
num_flowers = 5
```

- Floats - numbers with decimals

```
fraction = 0.2
```

- Booleans - true or false

```
is_raining_today = True
```

- Strings - collection of characters

```
myName = 'Ayca'
```

Called “doubles” in  
some other languages



Represented by bool



# Types

Suppose you're programming for a doctor's office...

What **type** would you use to store each of the following?



Think/Share

# Types

Suppose you're programming for a doctor's office...

What **type** would you use to store each of the following?

- The patient's weight
- The number of days since the patient's last visit
- The patient's temperature
- If the patient has had their flu shot
- The patient's number of children



# Types

Suppose you're programming for a doctor's office...

What **type** would you use to store each of the following?

- The patient's weight → **float**
- The number of days since the patient's last visit
- The patient's temperature
- If the patient has had their flu shot
- The patient's number of children



# Types

Suppose you're programming for a doctor's office...

What **type** would you use to store each of the following?

- The patient's weight → **float**
- The number of days since the patient's last visit → **integer**
- The patient's temperature
- If the patient has had their flu shot
- The patient's number of children



# Types

Suppose you're programming for a doctor's office...

What **type** would you use to store each of the following?

- The patient's weight → **float**
- The number of days since the patient's last visit → **integer**
- The patient's temperature → **float**
- If the patient has had their flu shot
- The patient's number of children



# Types

Suppose you're programming for a doctor's office...

What **type** would you use to store each of the following?

- The patient's weight → **float**
- The number of days since the patient's last visit → **integer**
- The patient's temperature → **float**
- If the patient has had their flu shot → **boolean**
- The patient's number of children



# Types

Suppose you're programming for a doctor's office...

What **type** would you use to store each of the following?

- The patient's weight → **float**
- The number of days since the patient's last visit → **integer**
- The patient's temperature → **float**
- If the patient has had their flu shot → **boolean**
- The patient's number of children → **integer**



# Explicit Type Conversion

```
num1 = 5  
num2 = 2  
num3 = 1.9
```

- Use **float(*value*)** to create new real-valued number

**float(num1)** = 5.0 (**float**)

- Note that **num1** is not changed. We created a new value.

**num1 + float(num2)** = 7.0 (**float**)

**num1 + num2** = 7 (**int**)

- Use **int(*value*)** to create a new integer-valued number  
(truncating anything after decimal)

**int(num3)** = 1 (**int**)

**int(-2.7)** = -2 (**int**)

# Explicit Type Conversion

```
num1 = 5  
num2 = 2  
num3 = 1.9
```

- Use **str**(*value*) to create new text out number

<b>float(num1)</b>	=	'5'	<b>(String)</b>
<b>float(num2)</b>	=	'2'	<b>(String)</b>
<b>float(num3)</b>	=	'1.9'.	<b>(String)</b>

Ready for another example?

# Another Program

```
def main():
    print("This program adds two numbers.")
```

# Recall, Our Program

```
def main():
    print("This program adds two numbers.")
```

This program adds two numbers.

- **print** command is displaying a **string**

# Recall, Our Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
```

num1      "9"

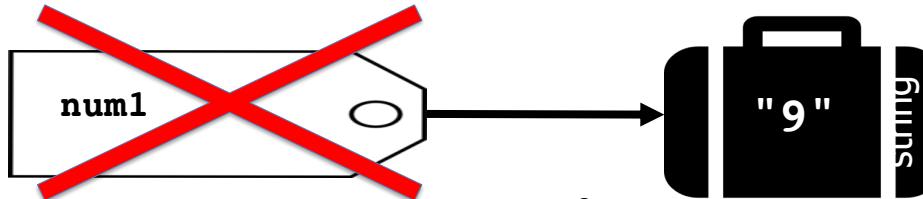
```
This program adds two numbers.
Enter first number: 9
```

- **input** command gives you back a **string**
  - Even if the user types in a number

# Show Me The Luggage!

- **input** command gives you back a **string**

```
num1 = input("Enter first number: ")
```

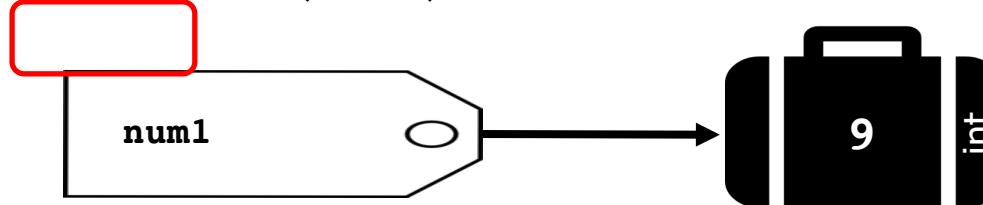


- We create an integer version of **num1**

```
num1 = int(num1)
```

- Create a new suitcase that has **int** version of **num1**
- Then assign the tag num1 to that piece of luggage

```
num1 = int(num1)
```



# Recall, Our Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
```

num1      "9"

This program adds two numbers.

Enter first number: 9

- Create **int** version of **string** and assign it back to **num1**

# Recall, Our Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
    num2 = input("Enter second number: ")
```

num1	9	num2	"17"
------	---	------	------

This program adds two numbers.

Enter first number: 9

Enter second number:

# Recall Our Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
    num2 = input("Enter second number: ")
    num2 = int(num2)
```

num1

9

num2

17

This program adds two numbers.

Enter first number: 9

Enter second number: 17

# Recall Our Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
    num2 = input("Enter second number: ")
    num2 = int(num2)
    total = num1 + num2
```

num1

9

num2

17

total

26

This program adds two numbers.

Enter first number: 9

Enter second number: 17

# Recall Our Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
    num2 = input("Enter second number: ")
    num2 = int(num2)
    total = num1 + num2
    print("The total is " + str(total) + ".")
```

num1

9

num2

17

total

26

This program adds two numbers.

Enter first number: 9

Enter second number: 17

The total is 26.

# What's Going on With `print`

- Adding strings in `print` command

```
print("The total is " + str(total) + ".")
```

- The `+` operator concatenates strings together

```
str1 = "hi"
```

```
str2 = " "
```

```
str3 = "there"
```

```
str4 = str1 + str2 + str3
```

- `total` is integer, so we need to create a string version

```
str(total)
```

- String version of `total` is a new value that is concatenated to produce final string that is printed
- Original variable `total` is still an `int`

# Recall Our Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
    num2 = input("Enter second number: ")
    num2 = int(num2)
    total = num1 + num2
    print("The total is " + str(total) + ".")
```

num1

9

num2

17

total

26

This program adds two numbers.

Enter first number: 9

Enter second number: 17

The total is 26.

# Side note about `print`

- You can `print` numbers by themselves directly
  - Only need to create string version of numbers when printing other text (strings) with them

```
def main():
    x = 10
    y = 3.5
    print(x)
    print(y)
    print("x = " + str(x))
```

```
10
3.5
x = 10
```

You just wrote your first  
Python program and learned  
about variables!

How do we process the information that we've stored?

# Expressions

## Recall: expressions

- In Karel, we only saw “boolean expressions” that evaluate to true/false
- In Python, expressions can evaluate to any type!
- The computer **evaluates** expressions to a single value
- We use **operators** to combine literals and/or variables into **expressions**

# Recall: expressions

- In Karel, we only saw “boolean expressions” that evaluate to true/false
  - In Python, expressions can evaluate to any type!
  - The computer **evaluates** expressions to a single value.
  - We use **operators** to combine literals and/or variables into **expressions**
-  Literals are Python objects written directly in code, e.g. the 5 in `num_flowers = 5`

# Performing Calculations

- **Math expression:** performs calculation and gives a value
  - Math operator: built-in tool for performing calculation
  - Operands: values surrounding operator
    - Variables can be used as operands
  - Resulting value typically assigned to variable

# Performing Calculations

## Arithmetic operators

- \* Multiplication
  - / Division
  - // Integer division
  - % Modulus (remainder)
  - + Addition
  - Subtraction
  - \*\* Exponentiation
- **Two types of division:**
    - / operator performs floating point division
    - // operator performs integer division
      - Positive results truncated, negative rounded away from zero

# Arithmetic Operators

```
num1 = 5  
num2 = 2
```

- Operations on numerical types (**int** and **float**)
- Operators

		<u>num3</u>
+	"addition"	Ex.: num3 = num1 + num2
-	"subtraction"	Ex.: num3 = num1 - num2
*	"multiplication"	Ex.: num3 = num1 * num2
/	"division"	Ex.: num3 = num1 / num2
//	"integer division"	Ex.: num3 = num1 // num2
%	"remainder"	Ex.: num3 = num1 % num2
**	"exponentiation"	Ex.: num3 = num1 ** num2
-	"negation" (unary)	Ex.: num3 = -num1

# Performing Calculations

## Arithmetic operators

*	Multiplication
/	Division
//	Integer division
%	Modulus (remainder)
+	Addition
-	Subtraction

Operator	Precedence
( )	1
* , / , // , %	2
+ , -	3

# Performing Calculations

## Arithmetic operators

*	Multiplication
/	Division
//	Integer division
%	Modulus (remainder)
+	Addition
-	Subtraction

Operator	Precedence
( )	1
* , / , // , %	2
+ , -	3

This is your “order of operations” for Python!

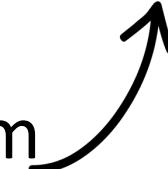
# Performing Calculations

## Arithmetic operators

*	Multiplication
/	Division
//	Integer division
%	Modulus (remainder)
+	Addition
-	Subtraction

Operator	Precedence
( )	1
* , / , // , %	2
+ , -	3

Ties within rows are  
broken by going from  
left to right



# Performing Calculations

Let's do some examples!

- $4 + 2 * 3$
- $5 + 1 / 2 - 4$
- $15 / 2.0 + 6$
- $5 + 1 / (2 - 4)$
- $5 + 1 // (2 - 4)$
- $1 * 2 + 3 * 5 \% 4$

Operator	Precedence
$()$	1
$\ast, /, //, \%$	2
$+, -$	3

Let's all think about it

# Performing Calculations

Let's do some examples!

- $4 + 2 * 3$
- $5 + 1 / 2 - 4$
- $15 / 2.0 + 6$
- $5 + 1 / (2 - 4)$
- $5 + 1 // (2 - 4)$
- $1 * 2 + 3 * 5 \% 4$

Operator	Precedence
$()$	1
$\ast, /, //, \%$	2
$+, -$	3

[demo]

# Performing Calculations

Let's do some examples!

- $4 + 2 * 3 \rightarrow 10$
- $5 + 1 / 2 - 4 \rightarrow 1.5$
- $15 / 2.0 + 6 \rightarrow 13.5$
- $5 + 1 / (2 - 4) \rightarrow 4.5$
- $5 + 1 // (2 - 4) \rightarrow 4$
- $1 * 2 + 3 * 5 \% 4 \rightarrow 5$

Operator	Precedence
$()$	1
$\ast, /, //, \%$	2
$+, -$	3

[demo]

# Performing Calculations

Let's do some examples!

- $4 + 2 * 3 \rightarrow 10$
- $5 + 1 / 2 - 4 \rightarrow 1.5$
- $15 / 2.0 + 6 \rightarrow 13.5$
- $5 + 1 / (2 - 4) \rightarrow 4.5$
- $5 + 1 // (2 - 4) \rightarrow 4$
- $1 * 2 + 3 * 5 \% 4 \rightarrow 5$

**NOTE:** Any of the literals can also be replaced with variables that are associated with the same value

# Performing Calculations

Let's do some examples!

- $4 + 2 * 3 \rightarrow 10$
- $5 + 1 / 2 - 4 \rightarrow 1.5$
- $15 / 2.0 + 6 \rightarrow 13.5$
- $5 + 1 / (2 - 4) \rightarrow 4.5$
- $5 + 1 // (2 - 4) \rightarrow 4$
- $1 * 2 + 3 * 5 \% 4 \rightarrow 5$

For example:

$$\begin{aligned}x &= 2 \\4 + x * 3\end{aligned}$$

This evaluates to 10, just like our first example expression!

# Expression Shorthands

```
num1 = 5  
num2 = 2  
num3 = 1.9
```

<b>num1</b> = <b>num1</b> + 1	same as	<b>num1</b> += 1
<b>num2</b> = <b>num2</b> - 4	same as	<b>num2</b> -= 4
<b>num3</b> = <b>num3</b> * 2	same as	<b>num3</b> *= 2
<b>num1</b> = <b>num1</b> / 2	same as	<b>num1</b> /= 2

- Generally:

***variable* = *variable* operator (*expression*)**

is same as:

***variable* operator= *expression***

# Implicit Type Conversion

```
num1 = 5  
num2 = 2  
num3 = 1.9
```

- Operations on two **ints** (except `/`) that would result in an integer value are of type **int**

`num1 + 7 = 12 (int)`

- Dividing `(/)` two **ints** results in a **float**, even if result is a round number  
(Ex.: `6 / 2 = 3.0`)

- If either (or both) of operands are **float**, the result is a **float**

`num3 + 1 = 2.9 (float)`

- Exponentiation depends on the result:

`num2 ** 3 = 8 (int)`

`2 ** -1 = 0.5 (float)`

How should we store information if it is known and never changes?

How should we store information if it is known and never changes?

Constants!

# Constants

Constants are like variables that don't change

- Constants give descriptive names to literals

## Style note

### **constants**

Use constants with descriptive names instead of literals directly in your code.

# Constants

Constants are like variables that don't change

- Constants give descriptive names to literals
- Use all capital letters and `snake_case` when naming constants

## Style note

### **constant names**

Use all capital letters and `snake_case`, for example `MY_CONSTANT = 500.`

# Constants

Constants are like variables that don't change

- Constants give descriptive names to literals
- Use all capital letters and `snake_case` when naming constants
- Constants are usually assigned outside functions and at the top of your program file (underneath the imports)

# Example of Using Constants

```
"""
File: constants.py
-----
An example program with constants
"""

INCHES_IN_FOOT = 12

def main():
    feet = float(input("Enter number of feet: "))
    inches = feet * INCHES_IN_FOOT
    print("That is " + str(inches) + " inches!")

# This provided line is required at the end of a Python file
# to call the main() function.
if __name__ == '__main__':
    main()
```

# Your job: Play with variables!

