Intro to Computer Science

Summer 2020

August 4th to August 20th, online

Variables and Expressions

Adopted from Stanford Uni's CS106ap course slides by Kylie Jue and Sonja Johnson-Yu and Code in Place by Piech and Sahami; Koca Uni's Comp125 course by Ayca Tuzmen

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

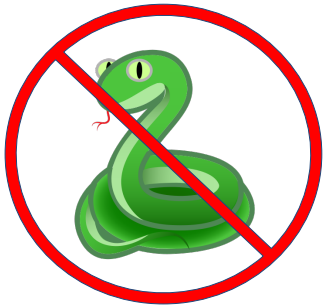
1. Welcome to Python
    - Input, output, process
2. Variables
    - Assignment and retrieval
    - Types
3. Using variables
    - In expressions

Welcome to Python

Welcome to Python

Guido van Rossum
(Creator of Python)

Monty Python's Flying Circus



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**

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

Input, Process, Output

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

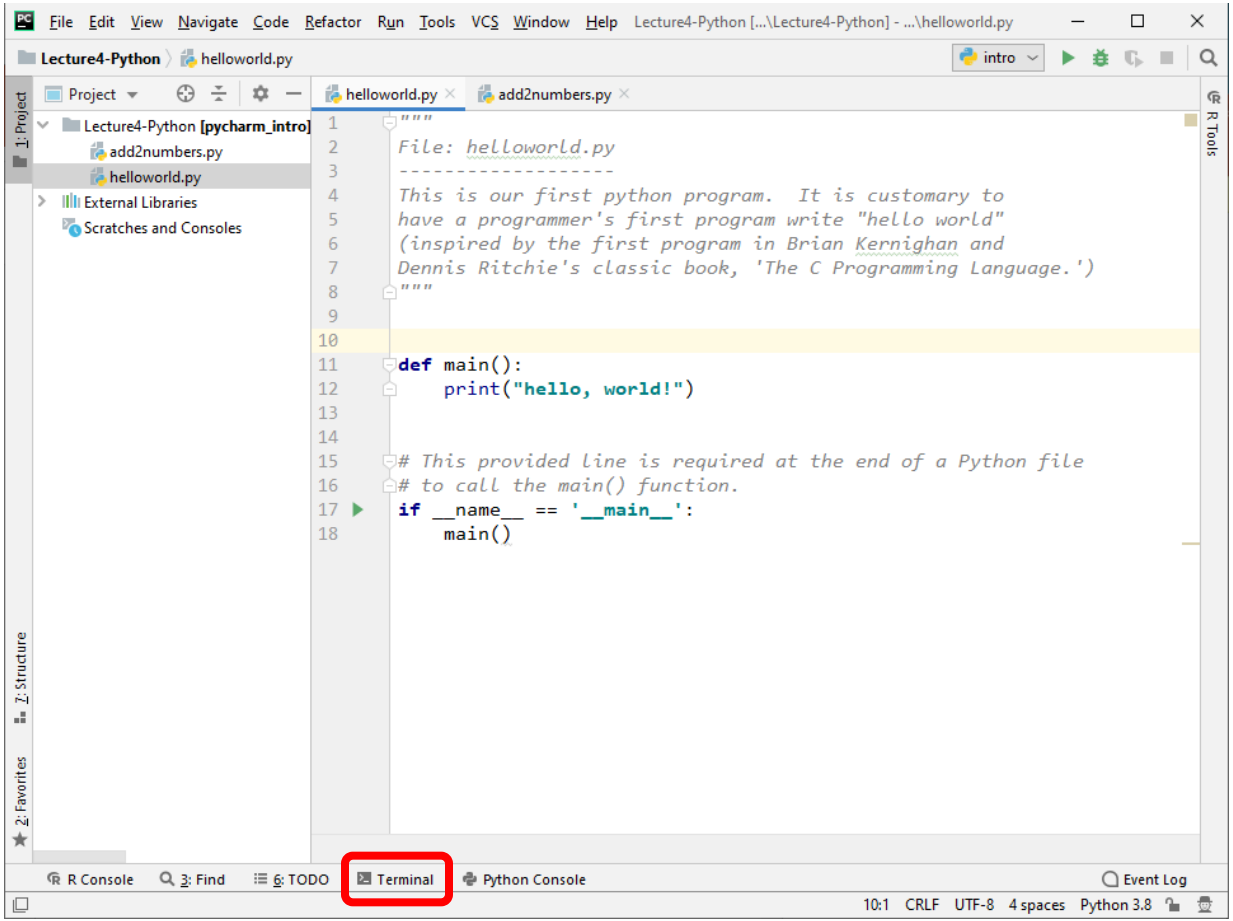
      `print('say "hi" Karel')` → `say "hi" Karel`

Our first program

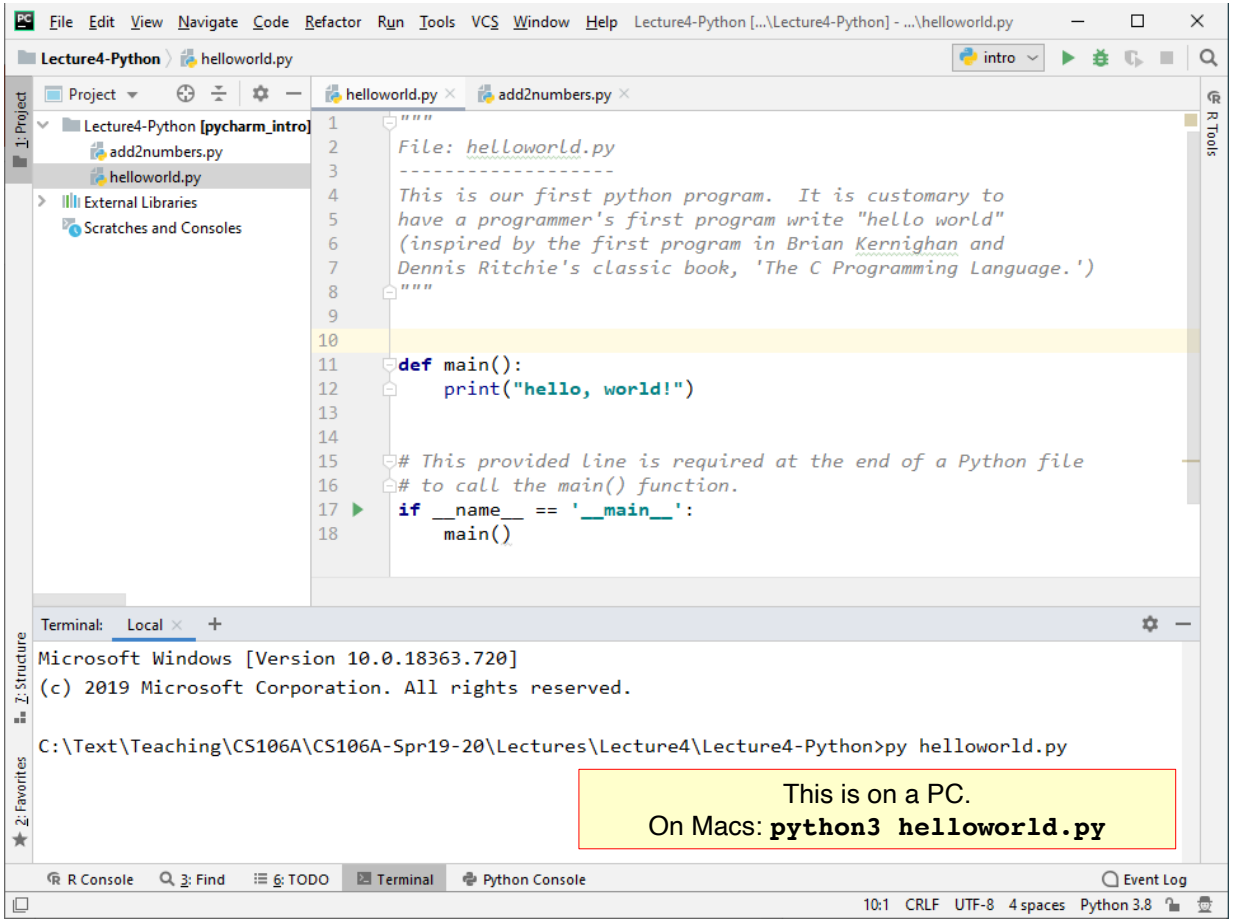
Our First Python Program

```
"""
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()                    # little bit different than in Karel
```

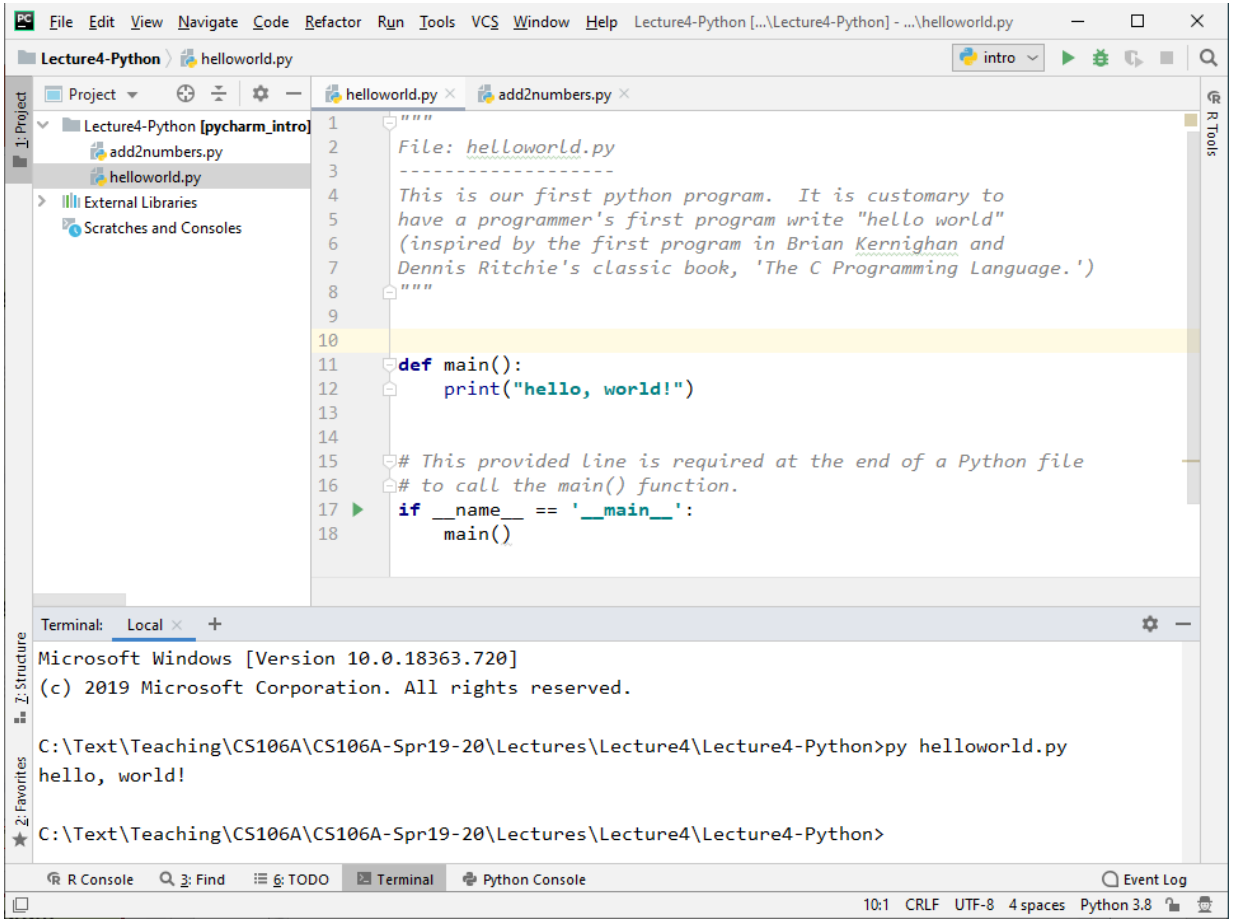
Our First Python Program



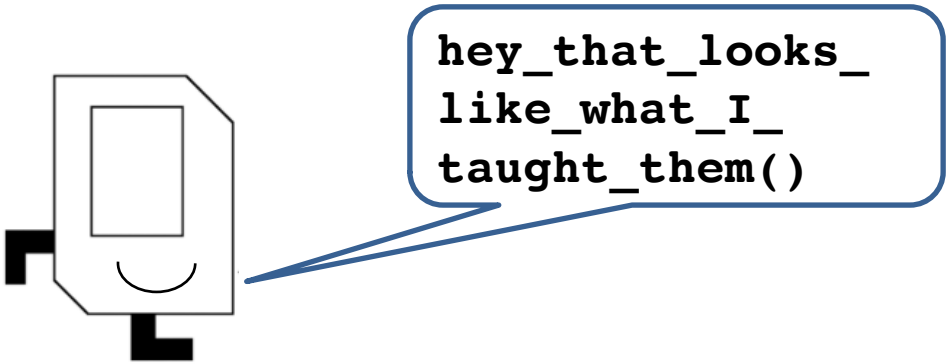
Our First Python Program



Our First Python Program



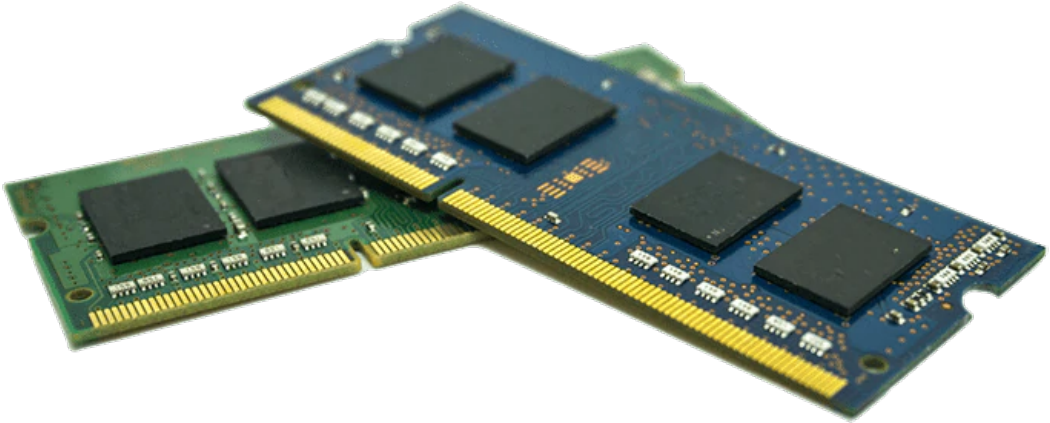
You're now all Python programmers!



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!

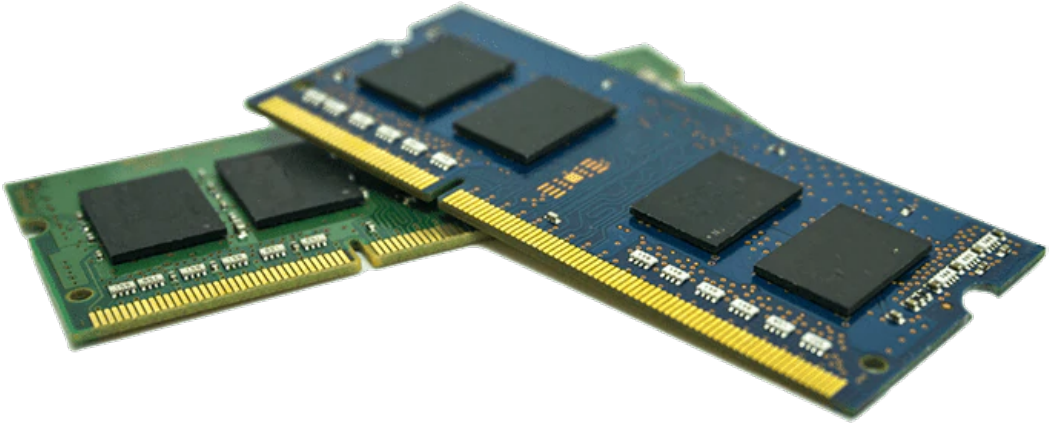
Definition

variable

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

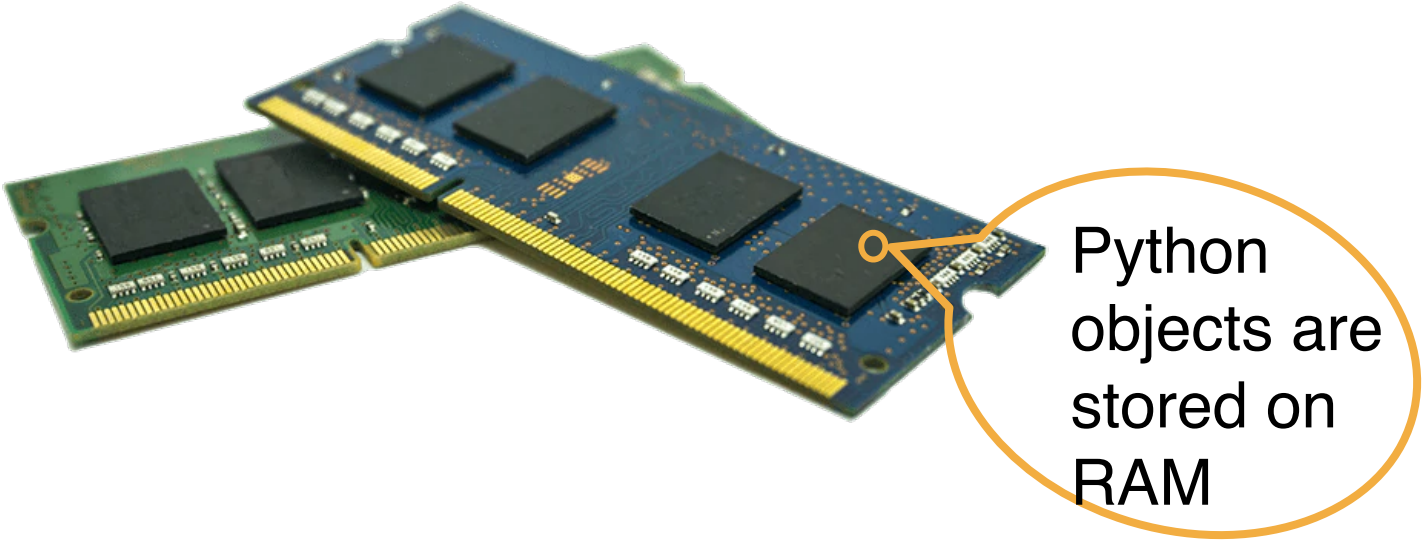
Suitcase Analogy

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



Suitcase Analogy

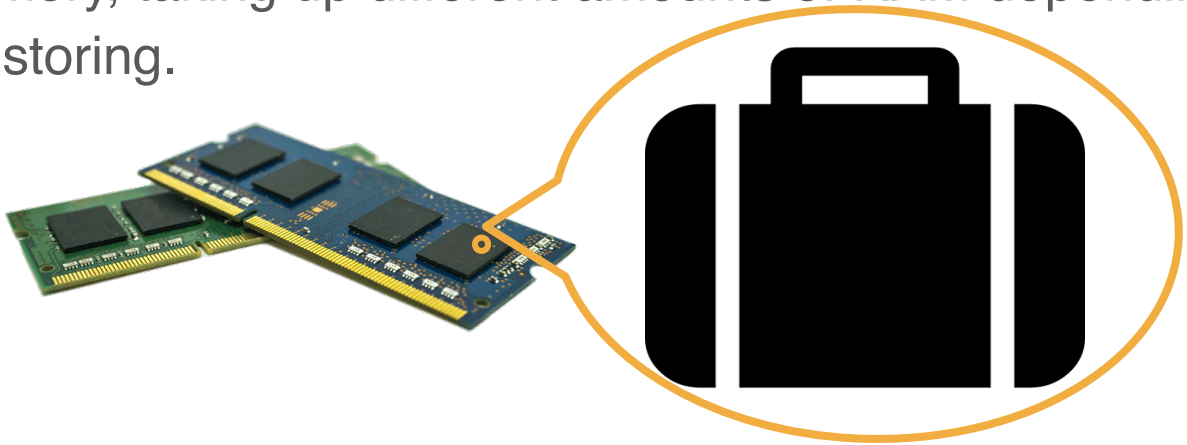
- When you store information in Python, it becomes a Python **object**
  - Objects come in different sizes and types (more on types later)



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

- 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.

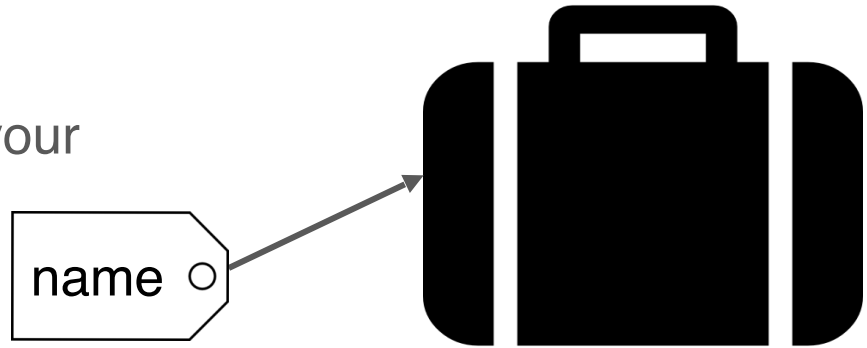
- A variable is a luggage tag for your suitcase that gives it a name!

Suitcase Analogy

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

- 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!



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

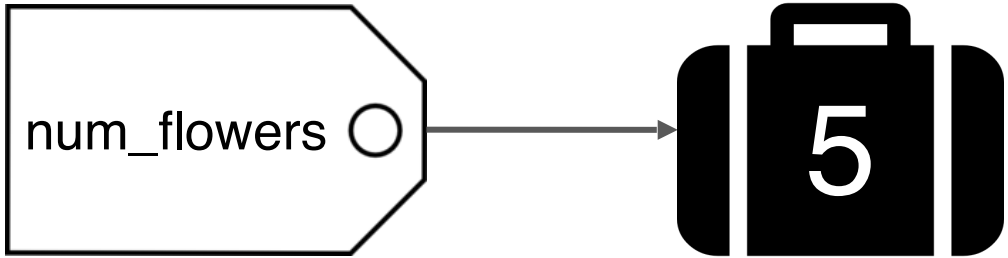
Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```

A Variable Example

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

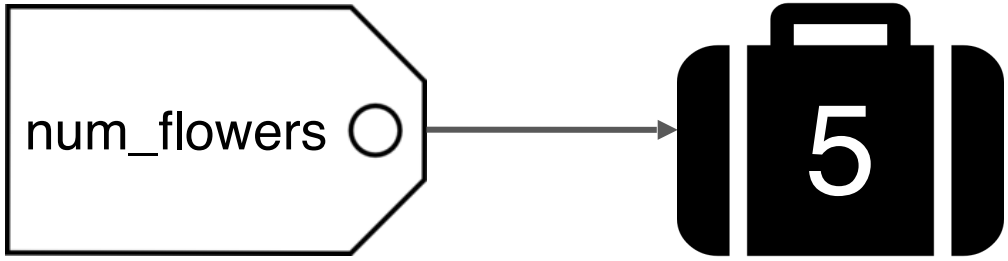
```
num_flowers = 5
```



A Variable Example

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

```
num_flowers = 5
```



Definition

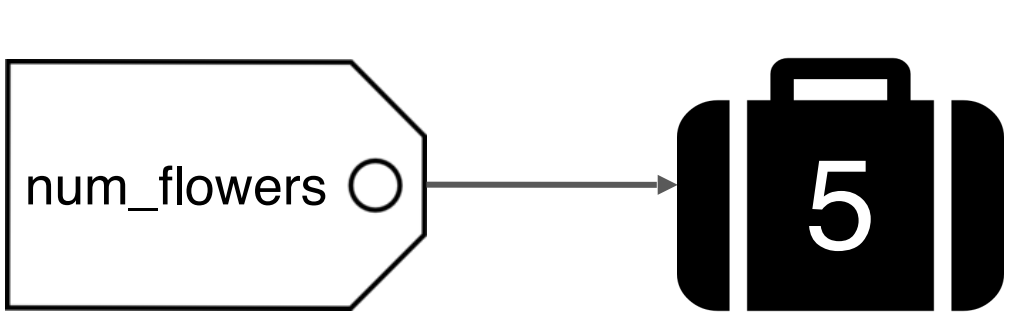
variable assignment

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

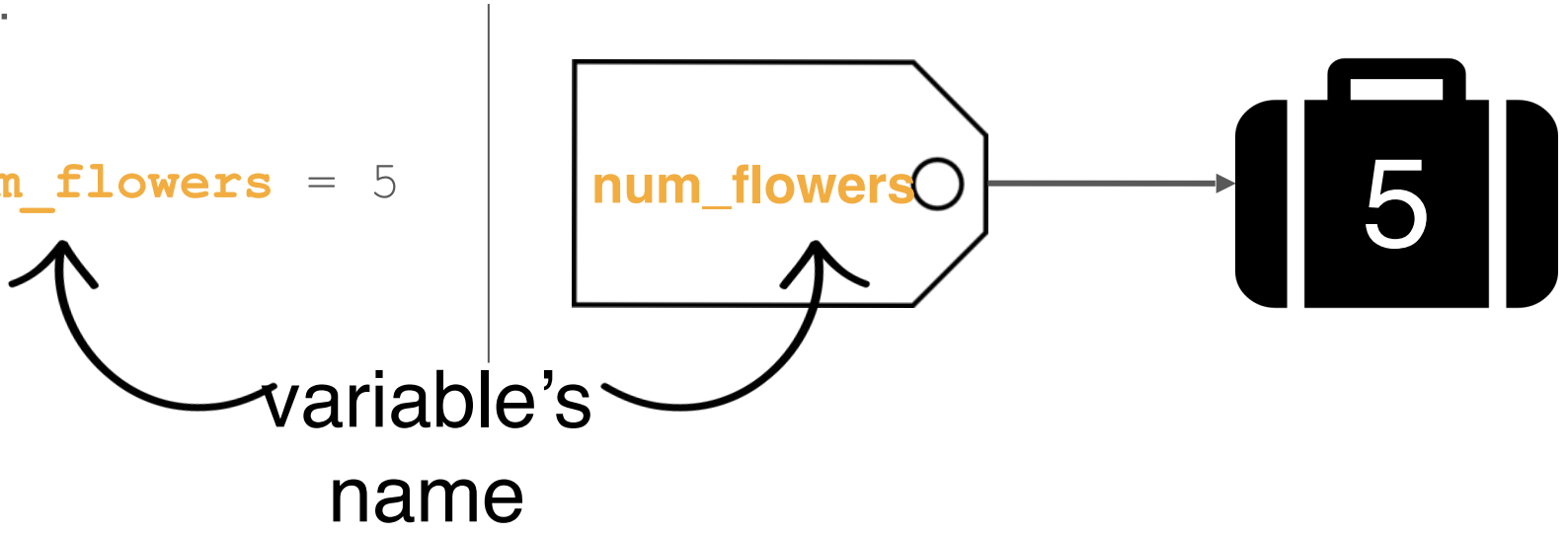
variable assignment

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:

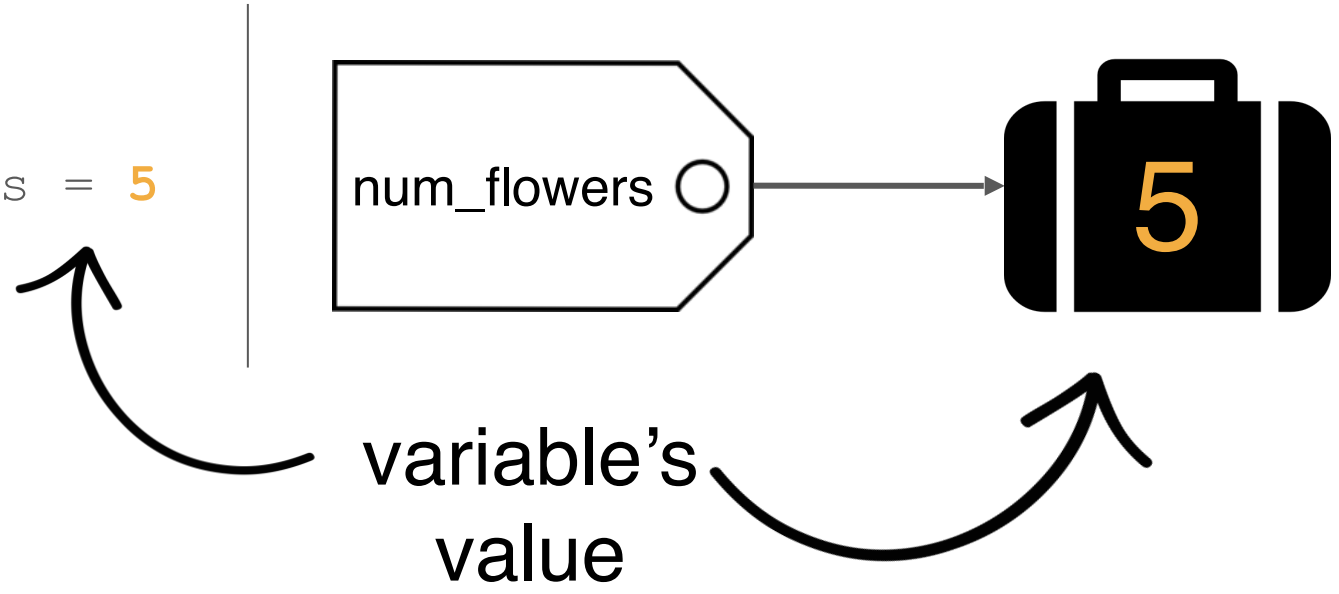
```
num_flowers = 5
```



A Variable Example

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

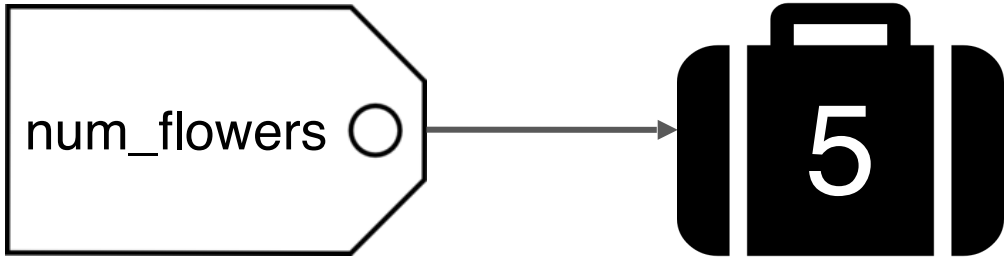
```
num_flowers = 5
```



A Variable Example

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

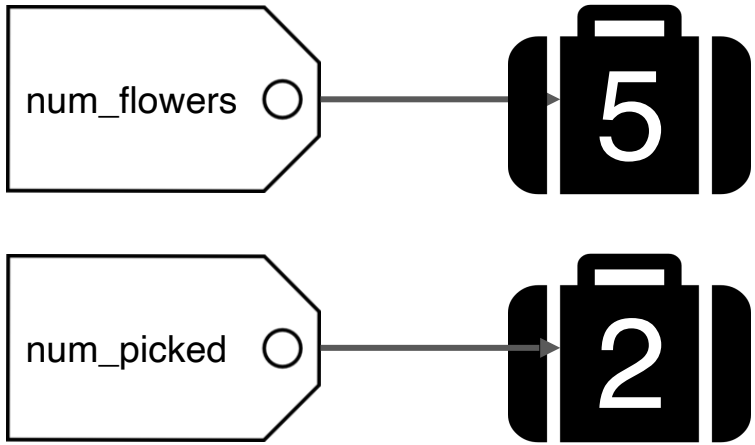
```
num_flowers = 5
```



A Variable Example

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

```
num_flowers = 5
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
```

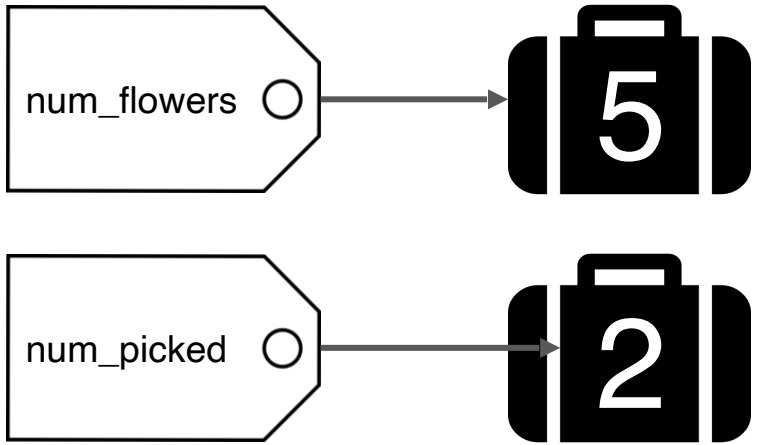
Think/Share: Try to predict what happens here!

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
```

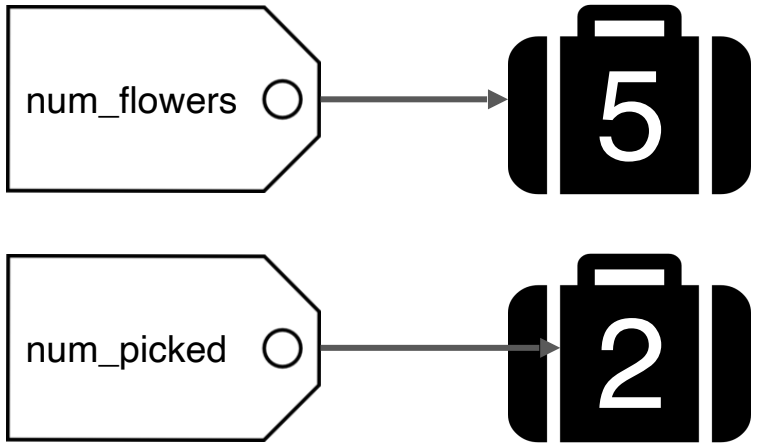
variable assignment!



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
```



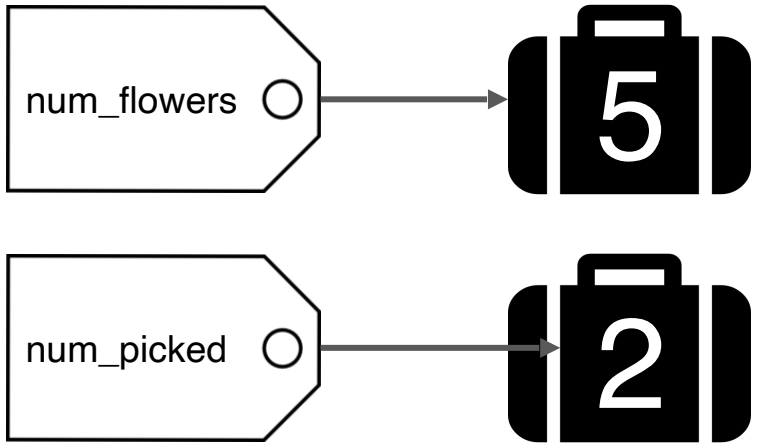
The right side of the equals sign **always** gets evaluated first.

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
```

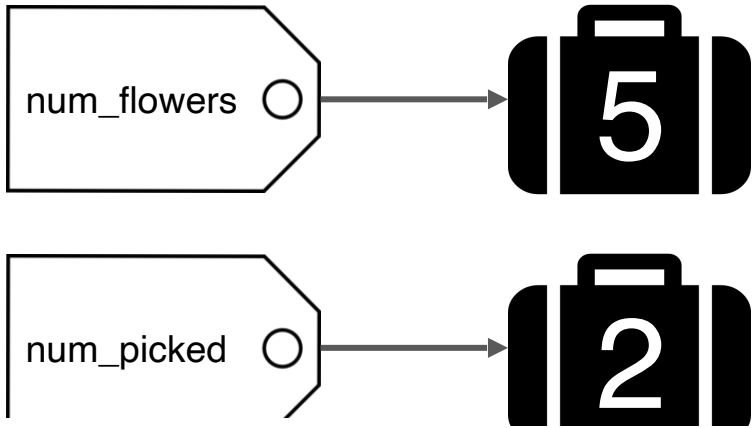
variable retrieval!



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

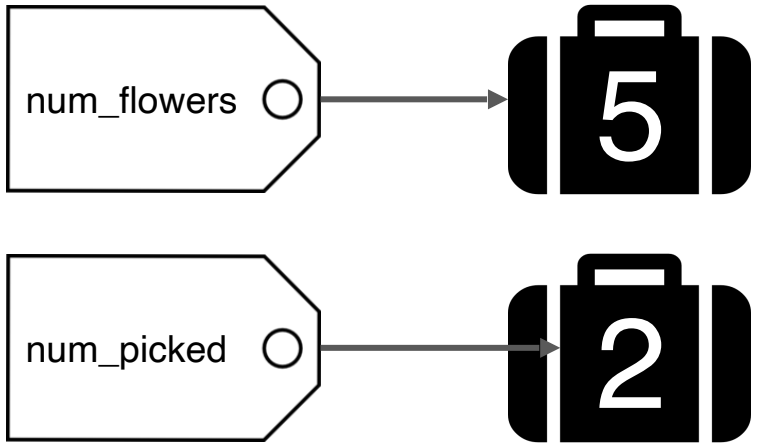
variable retrieval

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
```



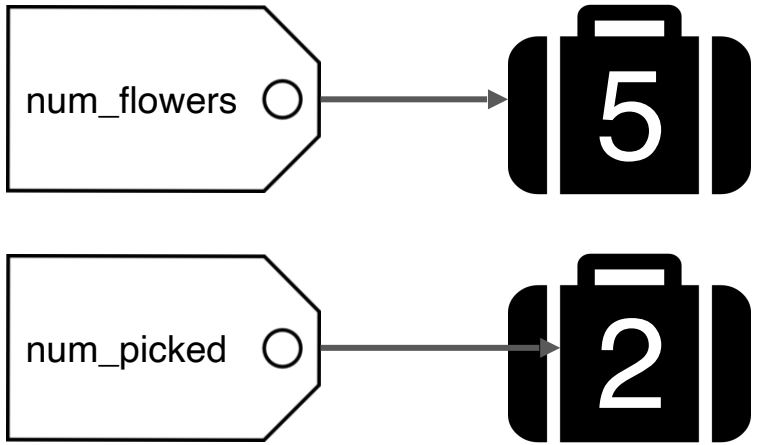
The right side of the equals sign **always** gets evaluated first.

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
```

5 2



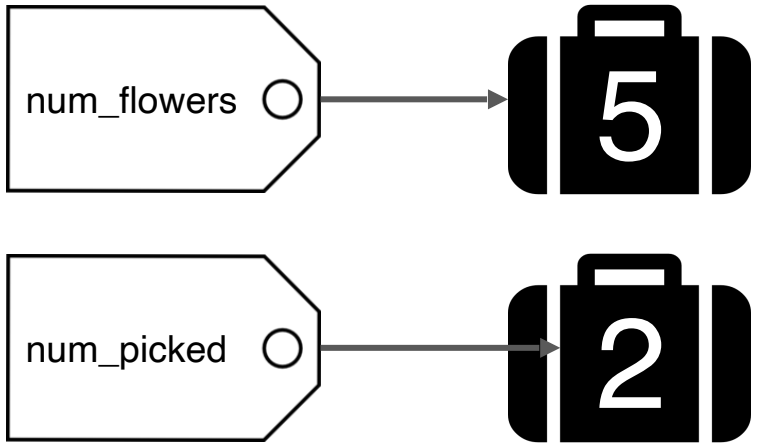
We get the values using variable retrieval (i.e. checking what suitcase is attached).

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
```

5 2



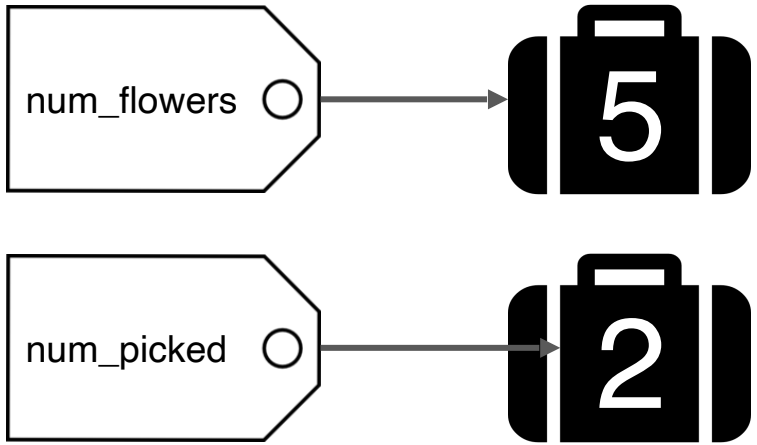
Then we can evaluate the right hand side of the assignment.

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
```

5 2 3



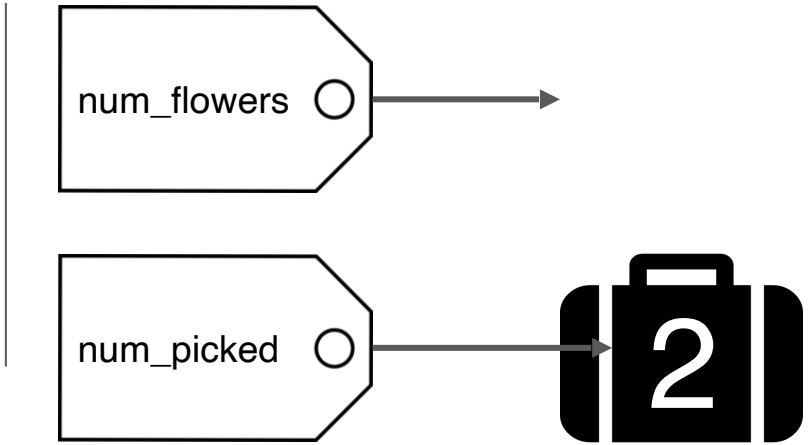
Then we can evaluate the right hand side of the assignment.

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



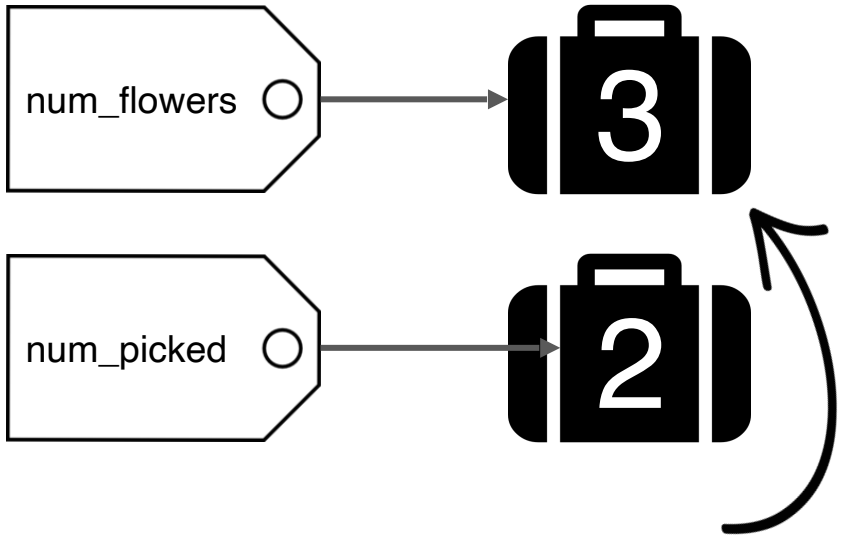
The right side of the equals sign **always** gets evaluated first.

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



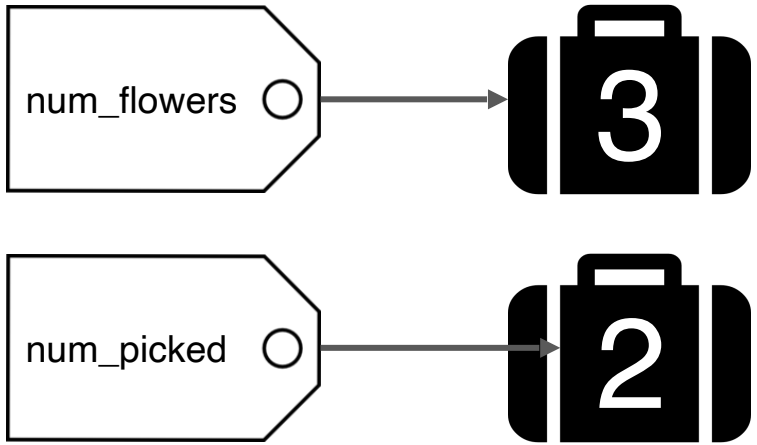
This is a new Python object!

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

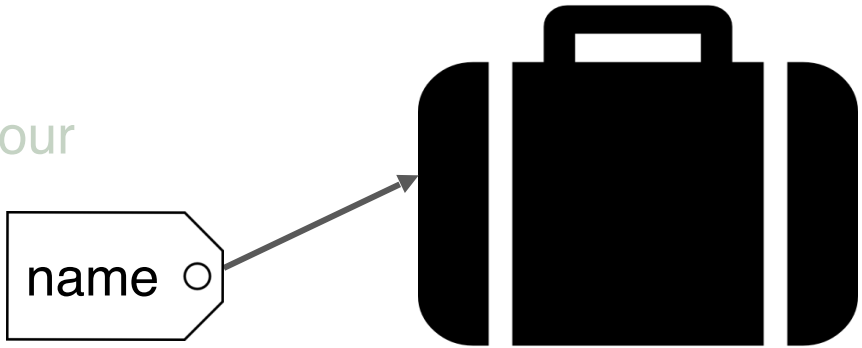
Data Types

The suitcase analogy

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

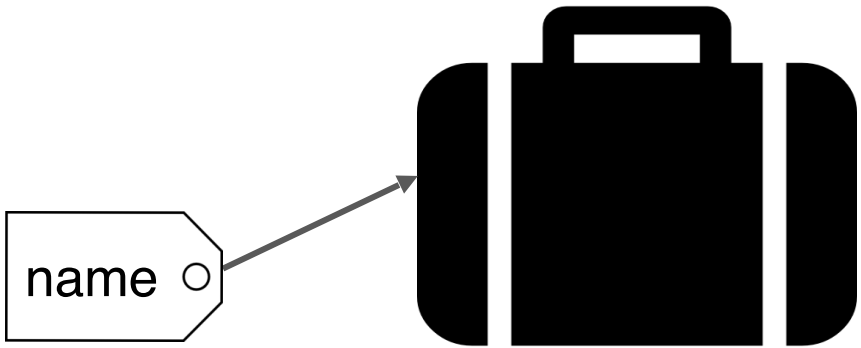
- You can think about a Python object as a suitcase stored in your computer's memory.

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



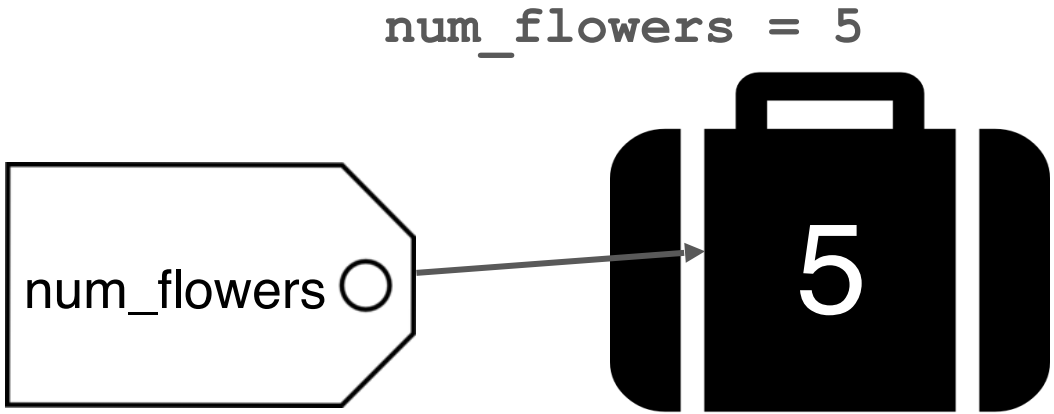
The suitcase analogy

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



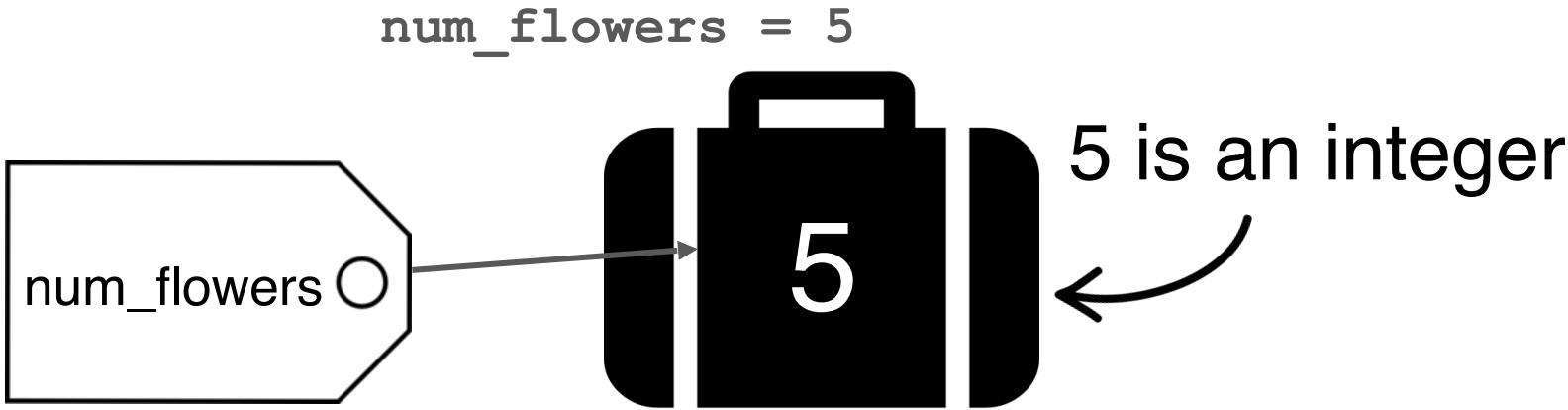
The suitcase analogy

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



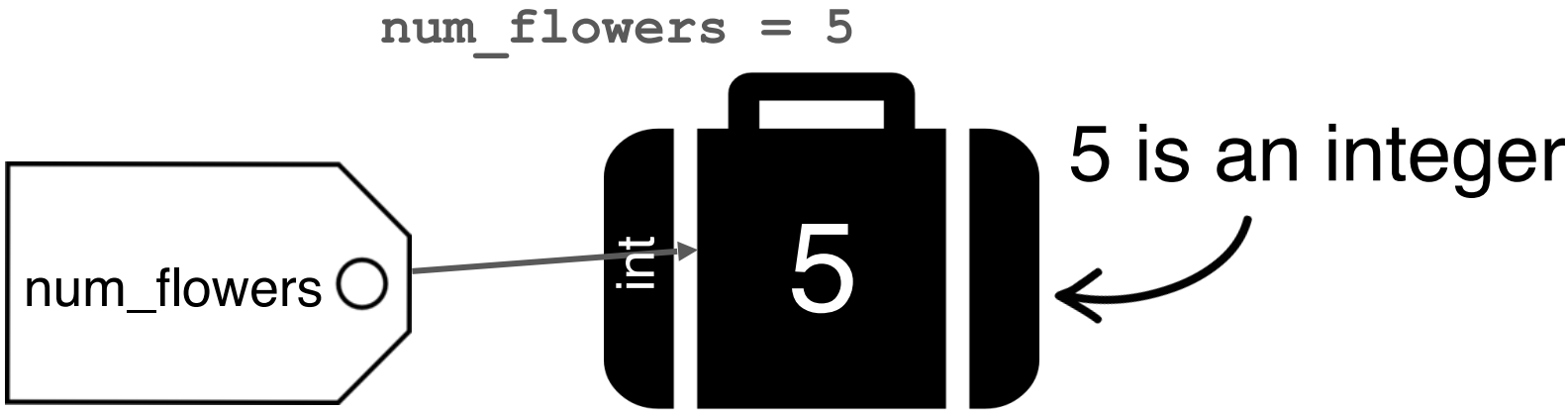
The suitcase analogy

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



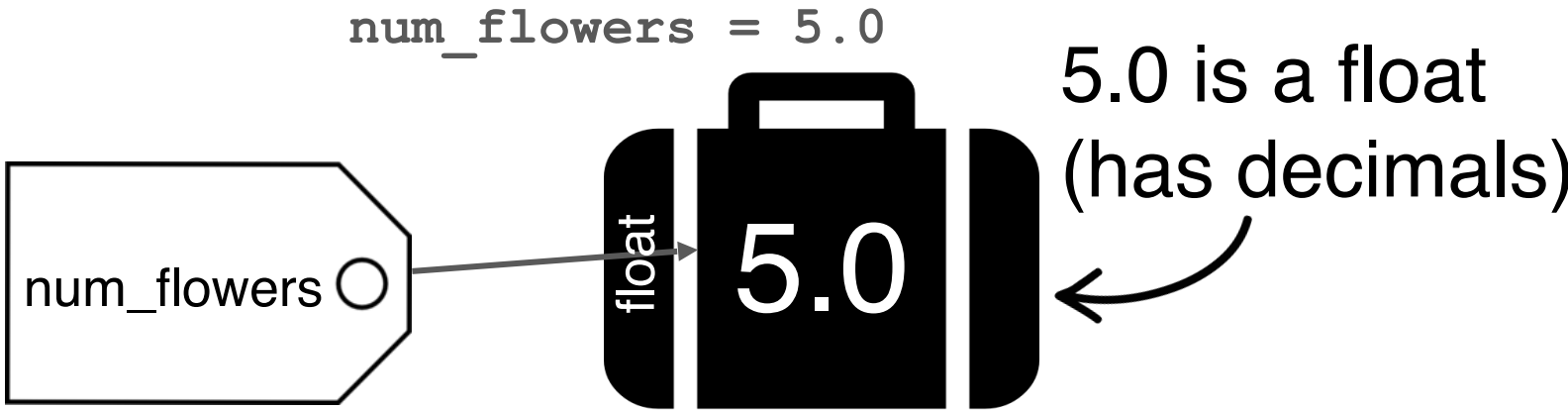
The suitcase analogy

- 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!

- 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

- Today we’ll cover about three different types:
  - Integers - numbers with no decimals

```
num_flowers = 5
```

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

- Today we'll cover about three different types:
  - Integers - numbers with no decimals
  - Floats - numbers with decimals

```
fraction = 0.2
```

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

- Today we'll cover about different types:
  - Integers - numbers with no decimals
  - Floats - numbers with decimals

```
fraction = 0.2
```

Called "doubles" in some other languages

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

- Today we'll cover about three different types:
  - Integers - numbers with no decimals
  - Floats - numbers with decimals
  - Booleans - true or false

```
is_raining_today = True
```

Types

- **Boolean variable: references one of two values, `True` or `False`**
  - Represented by `bool` data type
- **Commonly used as flags**
  - **Flag:** variable that signals when some condition exists in a program
    - Flag set to `False` → condition does not exist
    - Flag set to `True` → condition exists

Types

- **String: sequence of characters that is used as data**
- **String literal: string that appears in actual code of a program**
  - Must be enclosed in single (') or double (") quote marks
  - String literal can be enclosed in triple quotes (''' or """)
    - Enclosed string can contain both single and double quotes and can have multiple lines

Types

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

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

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

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 → **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...

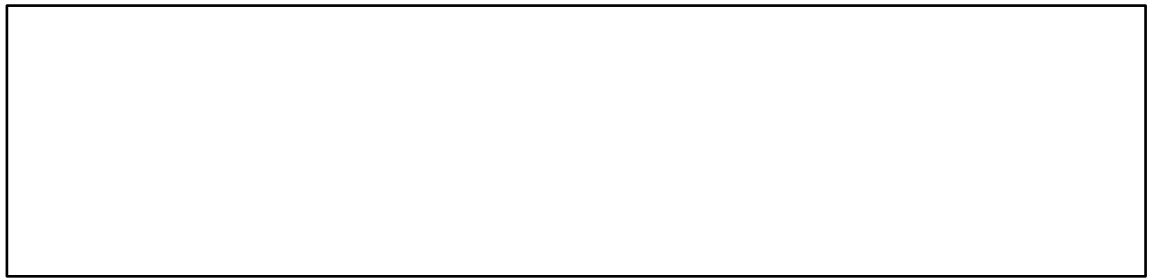
What **type** would you use to store each of the following?

- 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**

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

Recall, Our Program

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

num1 | "9"

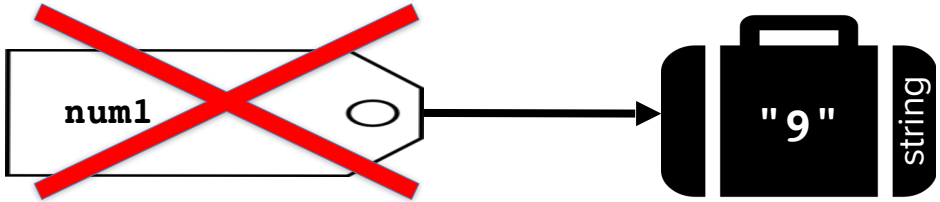
```
This program adds two numbers.
Enter first number: 9
```

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

Show Me The Luggage!

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

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

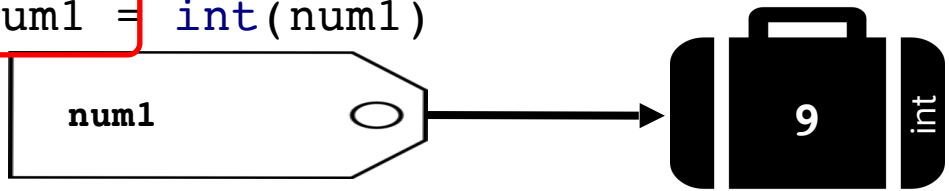


  - We create an integer version of **num1**

```
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)
    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: 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

			num3
`+`	"addition"	Ex.: num3 = `num1 + num2`	7
`-`	"subtraction"	Ex.: num3 = `num1 - num2`	3
`*`	"multiplication"	Ex.: num3 = `num1 * num2`	10
`/`	"division"	Ex.: num3 = `num1 / num2`	2.5
`//`	"integer division"	Ex.: num3 = `num1 // num2`	2
`%`	"remainder"	Ex.: num3 = `num1 % num2`	1
`**`	"exponentiation"	Ex.: num3 = `num1 ** num2`	25
`-`	"negation" (unary)	Ex.: num3 = `-num1`	-5

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
`*, /, //, %`	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
`*, /, //, %`	2
`+, -`	3

[demo]

Performing Calculations

Let's do some examples!

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

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

[demo]

Performing Calculations

Let's do some examples!

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

For example:

```
x = 2
4 + x * 3
```

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

Converting Math Formulas to Programming Statements

- **Operator required for any mathematical operation**
- **When converting mathematical expression to programming statement:**
  - May need to add multiplication operators
  - May need to insert parentheses

Converting Math Formulas to Programming Statements

Algebraic Expression	Python Statement
$y = 3\frac{x}{2}$	`y = 3 * x / 2`
$z = 3bc + 4$	`z = 3 * b * c + 4`
$a = \frac{x + 2}{b - 1}$	`a = (x + 2) / (b - 1)`

Expression Shorthands

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

num1 = num1 + 1	same as	**num1 += 1**
num2 = num2 - 4	same as	**num2 -= 4**
num3 = num3 * 2	same as	**num3 *= 2**
num1 = num1 / 2	same as	**num1 /= 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 **int**s (except **/**) that would result in an integer value are of type **int**

  num1 + 7 = 12 (int)

  – Dividing (**/**) two **int**s 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)

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)`**

What if the operator is not built in?

Python `math` Library

```
import math
```

- math library has many built-in constants:

`math.pi`	mathematical constant π
`math.e`	mathematical constant e

- and useful functions:

`math.sqrt(`x`)`	returns square root of x
`math.exp(`x`)`	returns e^x
`math.log(`x`)`	returns natural log (base e) of x

- These are just a few examples of what's in math

Example of Using `math` Library

```
"""
File: squareroot.py
-------------------
This program computes square roots
"""

import math

def main():




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

Example of Using `math` Library

```
"""
File: squareroot.py
-------------------
This program computes square roots
"""

import math

def main():
    num = float(input("Enter number: "))



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

Example of Using `math` Library

```
"""
File: squareroot.py
-------------------
This program computes square roots
"""

import math

def main():
    num = float(input("Enter number: "))
    root = math.sqrt(num)


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

Example of Using `math` Library

```
"""
File: squareroot.py
-------------------
This program computes square roots
"""

import math

def main():
    num = float(input("Enter number: "))
    root = math.sqrt(num)
    print("Square root of " + str(num) + " is " + str(root))

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

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)

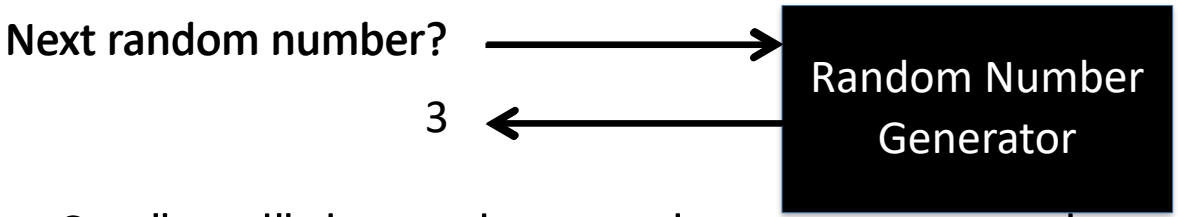
Example of Using Constants

```
"""
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()
```

What if we want to randomly generate data?

Random Number Generation

- Want a way to generate random number
  - Say, for games or other applications
- No "true" randomness in computer, so we have *pseudorandom* numbers
  - "That looks pretty random to me"
- Want "black box" that we can ask for random numbers



- Can "seed" the random number generator to always produce the same sequence of "random" numbers

Python `random` Library

```
import random
```

Function	What it does
`random.randint(`***min***`, `***max***`)`	Returns a random integer between *min* and *max*, inclusive.
`random.random()`	Returns a random real number (float) between 0 and 1.
`random.uniform(`***min***`, `***max***`)`	Returns a random real number (float) between *min* and *max*.
random.seed(***x***)	Sets "seed" of random number generator to *x*.

Let's consider an example
rolldice.py

Example of Using `random` Library

```
"""
File: rolldice.py
------------------
Simulate rolling two dice
"""

import random

NUM_SIDES = 6

def main():
```

Example of Using `random` Library

```
"""
File: rolldice.py
------------------
Simulate rolling two dice
"""

import random

NUM_SIDES = 6

def main():


    die1 = random.randint(1, NUM_SIDES)
```

Example of Using `random` Library

```
"""
File: rolldice.py
------------------
Simulate rolling two dice
"""

import random

NUM_SIDES = 6

def main():


    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
```

Example of Using `random` Library

```
"""
File: rolldice.py
------------------
Simulate rolling two dice
"""

import random

NUM_SIDES = 6

def main():


    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
```

Example of Using `random` Library

```
"""
File: rolldice.py
------------------
Simulate rolling two dice
"""

import random

NUM_SIDES = 6

def main():


    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print("Dice have " + str(NUM_SIDES) + " sides each.")
    print("First die: " + str(die1))
    print("Second die: " + str(die2))
    print("Total of two dice: " + str(total))
```

Example of Using `random` Library

```
"""
File: rolldice.py
------------------
Simulate rolling two dice
"""

import random

NUM_SIDES = 6

def main():
    # setting seed is useful for debugging
    # random.seed(1)
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print("Dice have " + str(NUM_SIDES) + " sides each.")
    print("First die: " + str(die1))
    print("Second die: " + str(die2))
    print("Total of two dice: " + str(total))
```

Your job: Play with variables!