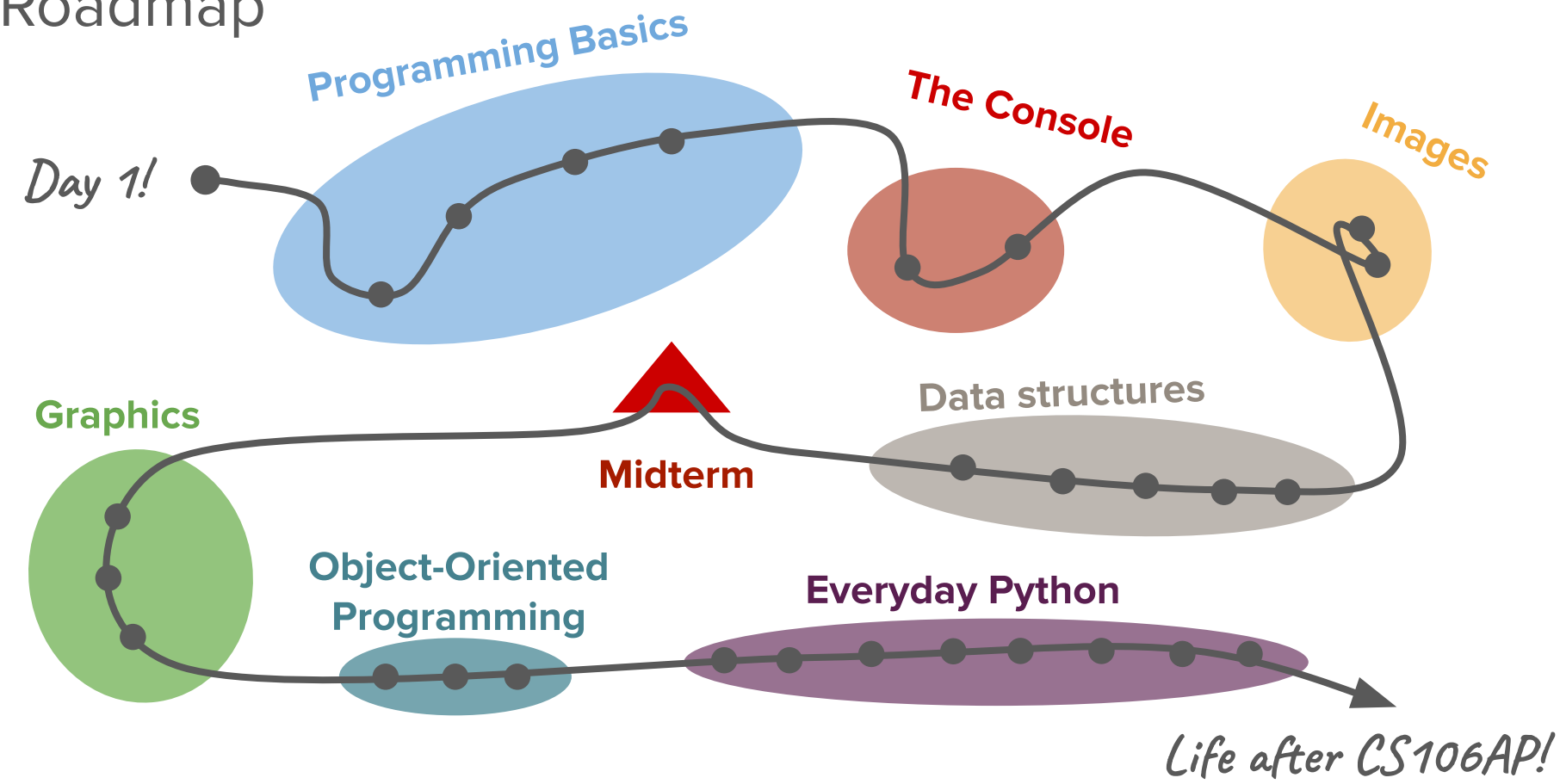
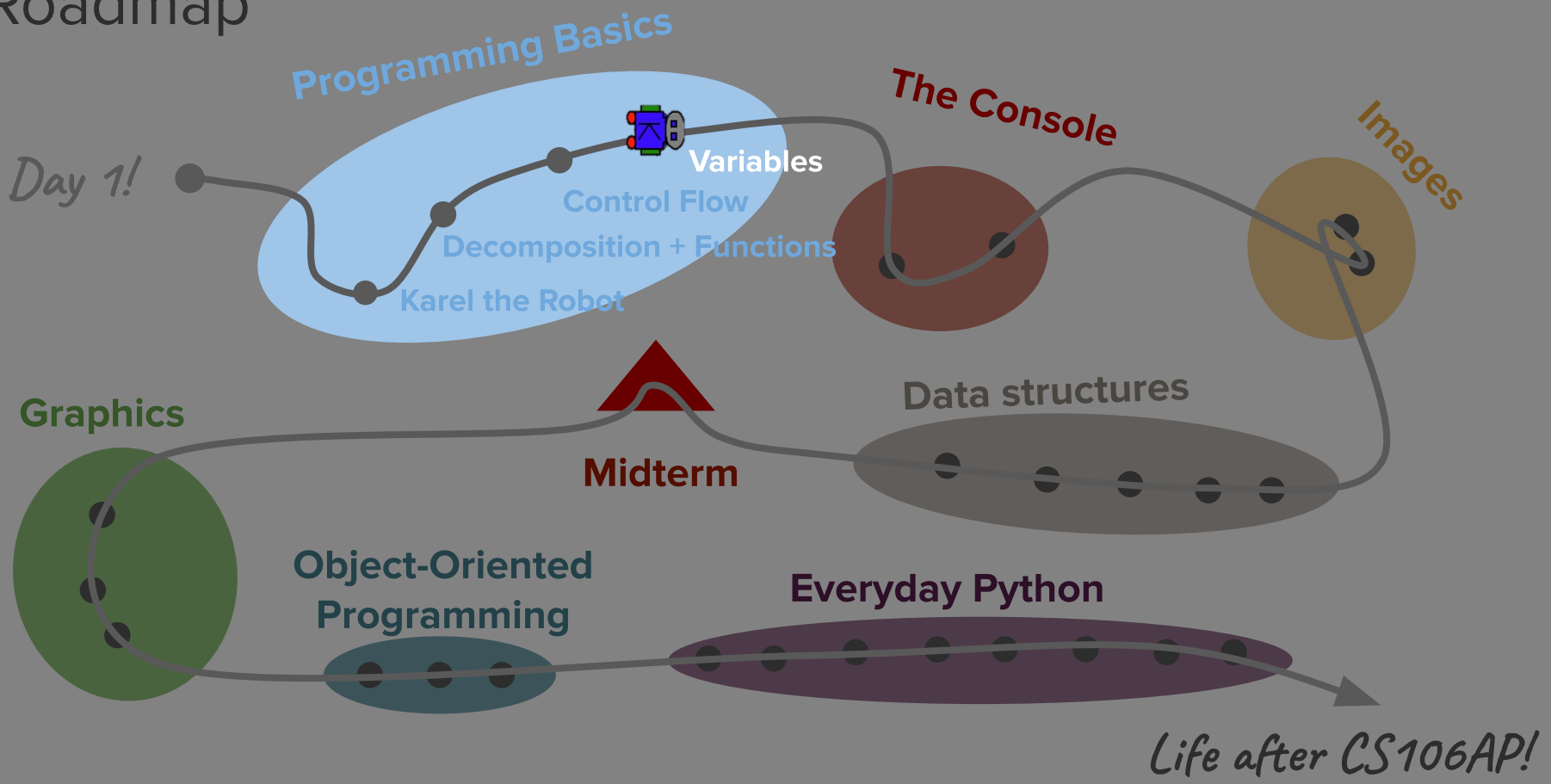
Variables and Expressions

CS106AP Lecture 5

Roadmap


Programming Basics
The Console
Images
Day 1!
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Roadmap


Programming Basics
Day 1!
Variables
Control Flow
Decomposition + Functions
Karel the Robot
The Console
Images
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Today’s questions

How do computers store information (data) using code?

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

Today's topics

1. Review
2. Variables
   - Assignment and retrieval
   - Types
3. Using variables
   - In expressions
   - In functions
4. Constants
5. What's next?

Review

Coding style tactics

A note about PEP 8

- PEP - Python Enhancement Proposal
- PEP 8 is an official standard for Python code formatting
- All the code we show you will follow PEP 8, and PyCharm also enforces it by default!
  - You should therefore follow this by default in your code.
- In real-world projects, some are super strict about following PEP 8.

What we've learned so far

- Have descriptive function names
- Decompose your programs
- Use consistent spacing (4 spaces per level of indentation!)
- Comment your code **as you write each function**

What we've learned so far

- Have descriptive function names
- Decompose your programs
- Use consistent spacing (4 spaces per level of indentation!)
- Comment your code **as you write each function**
- Avoid unnecessary control flow checks!

What we've learned so far

- Have descriptive function names
- Decompose your programs
- Use consistent spacing (4 spaces per level of indentation!)
- Comment your code **as you write each function**
- Avoid unnecessary control flow checks!

Ask yourself if all of the conditions always need to be checked!

Control flow 2.0

Combining boolean expressions: **`not`**, **`and`**, & **`or`**

- Python uses the words **`not`**, **`and`**, & **`or`** to combine boolean values
  - **`not`** `boolean_expression`
    Inverts True/False
  - `boolean_expression_a` **`and`** `boolean_expression_b`
    If both sides are true, the entire condition is true.
  - `boolean_expression_a` **`or`** `boolean_expression_b`
    If either side is true, the entire condition is true.

If-else and if-elif-else statements

```
if ____________:
    # Do something
else:
    # Do something else
```

```
if ____________:
    # Do something
elif ____________:
    # Do something different
else:
    # Do something else
```

If-elif-else statements

- Can have more than one elif block
  - But too many can get messy

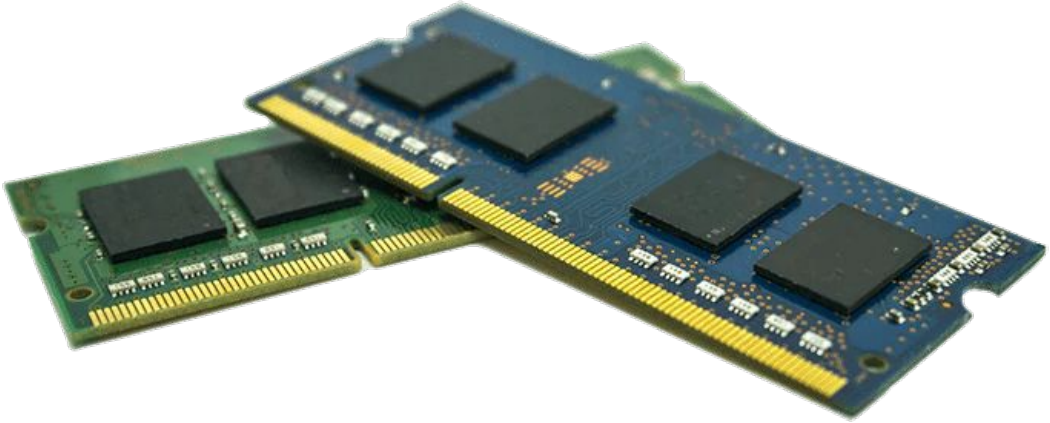
- Else is optional
  - You can think of this like a default option

- Put the conditions in priority order!
  - Ask: Which condition do I want to check first?

How do computers store information (data)?

Your computer has memory!

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



- We'll talk more about RAM and other parts of your computer later this quarter

Image (c) 2019 HP Development Company, L.P. [source]

How do computers store information (data) in code?

How do computers store information (data) in code?

Variables!

Definition

variable

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

Think of them as labels for containers!

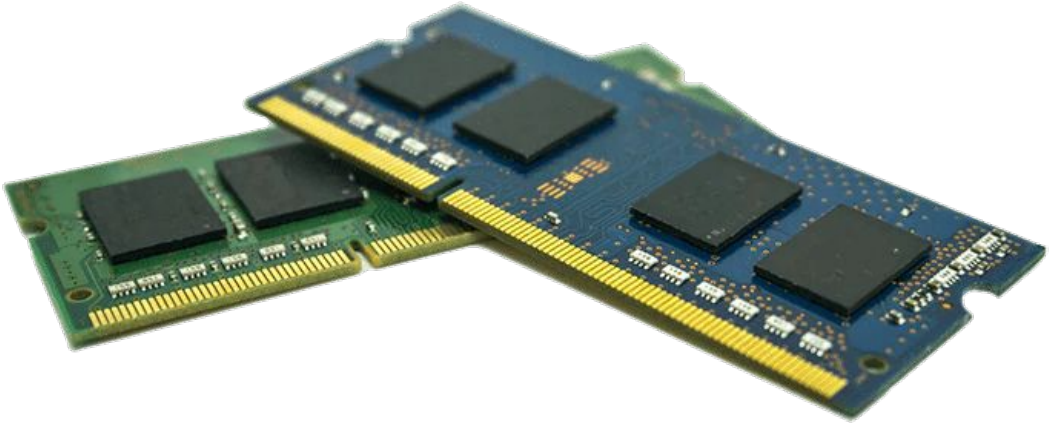
Definition

variable

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

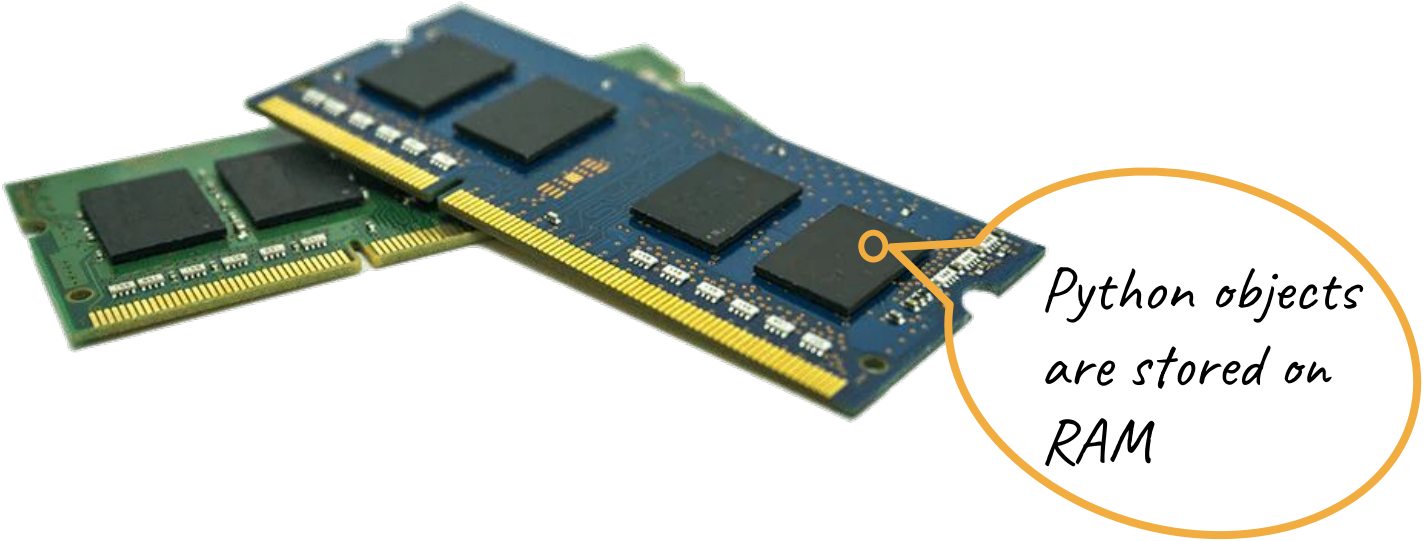
The suitcase analogy

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



The suitcase analogy

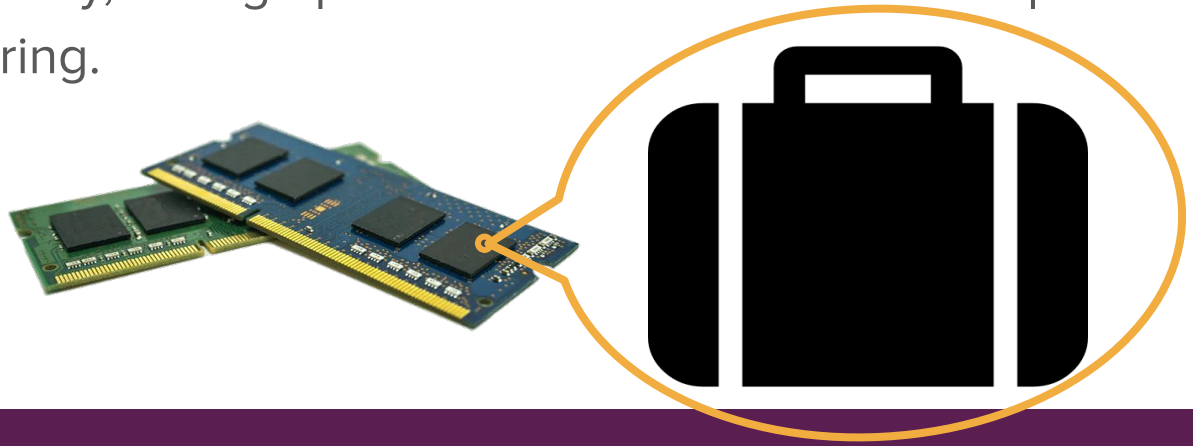
- When you store information in Python, it becomes a Python **object**
  - Objects come in different sizes and types (more on types later)



The 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.



The 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.

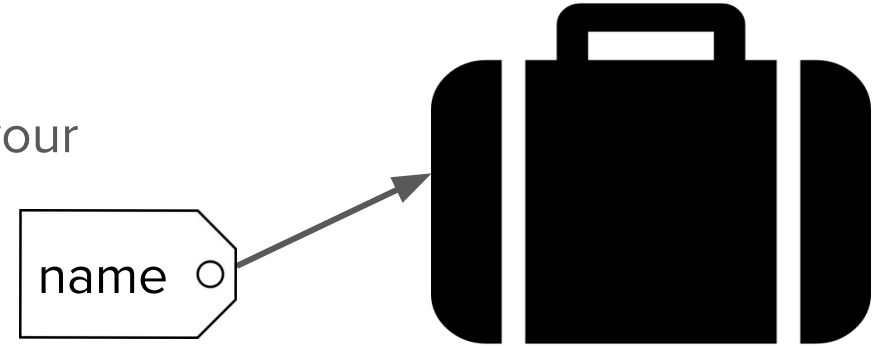
- A variable is a luggage tag for your suitcase that gives it a name!

The 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!



An example

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

An example

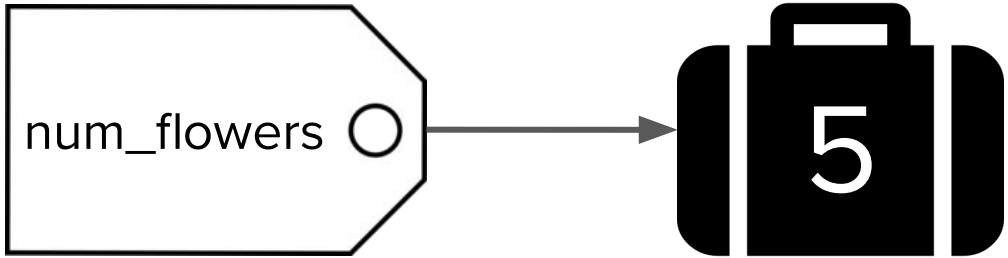
Suppose you're writing a program that keeps track of the flowers in your garden:

```
num_flowers = 5
```

An example

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

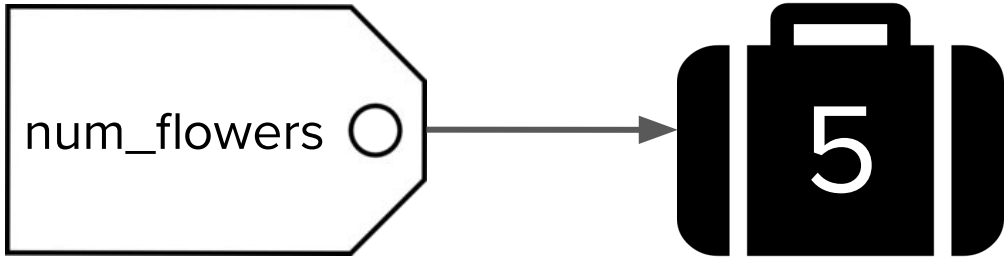
```
num_flowers = 5
```



An example

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

```
num_flowers = 5
```



Definition

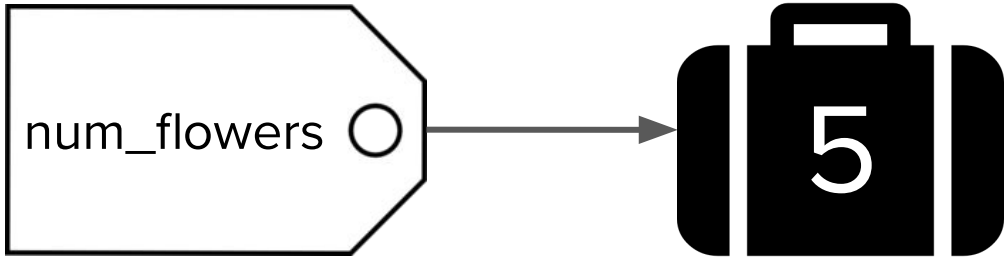
variable assignment

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

An 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

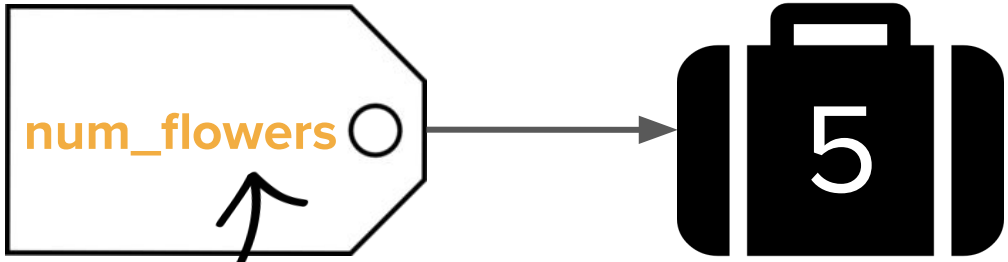
variable assignment

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

An example

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

```
num_flowers = 5
```

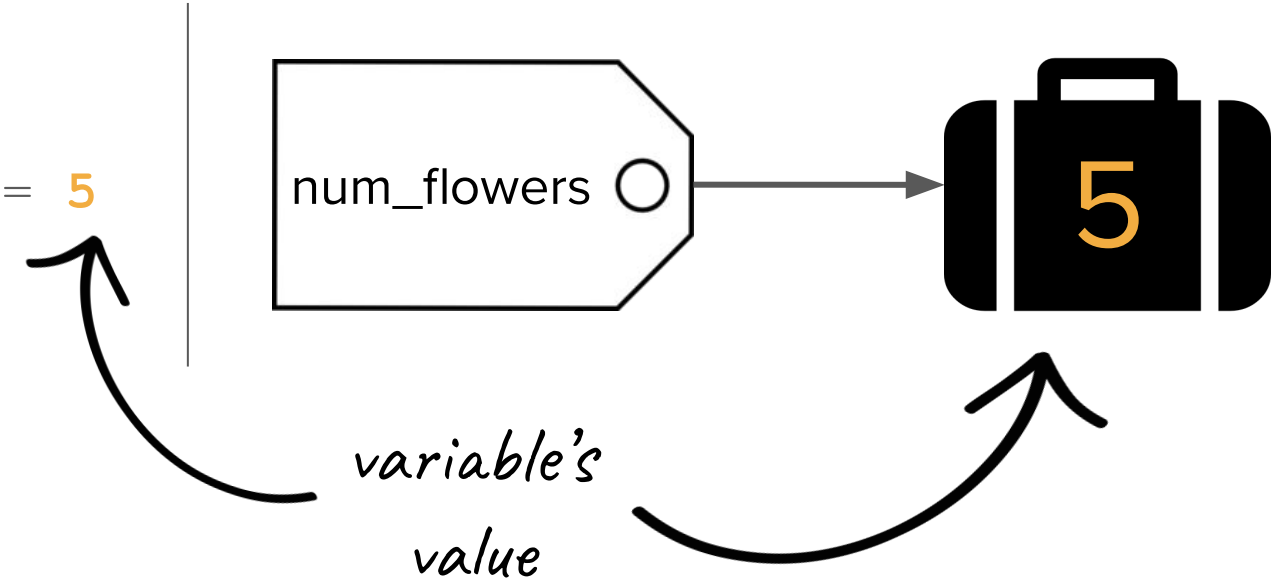


variable's name

An example

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

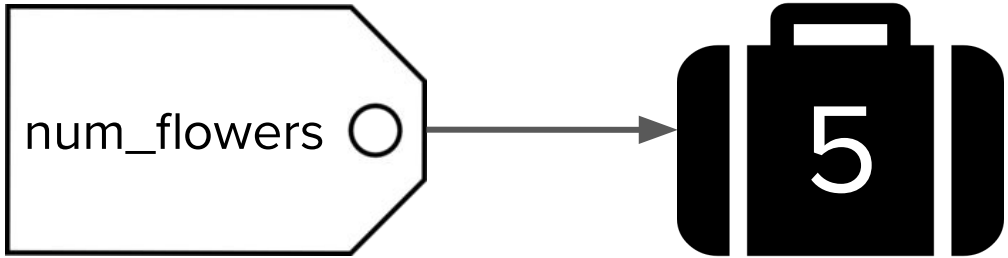
```
num_flowers = 5
```



An example

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

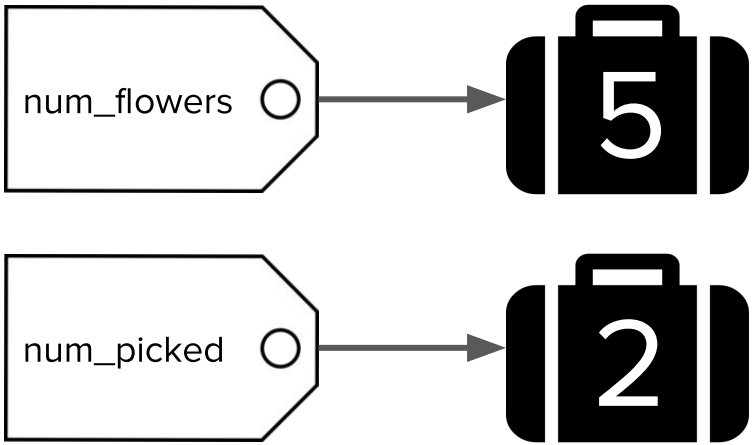
```
num_flowers = 5
```



An example

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

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



An 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
```

An 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
```

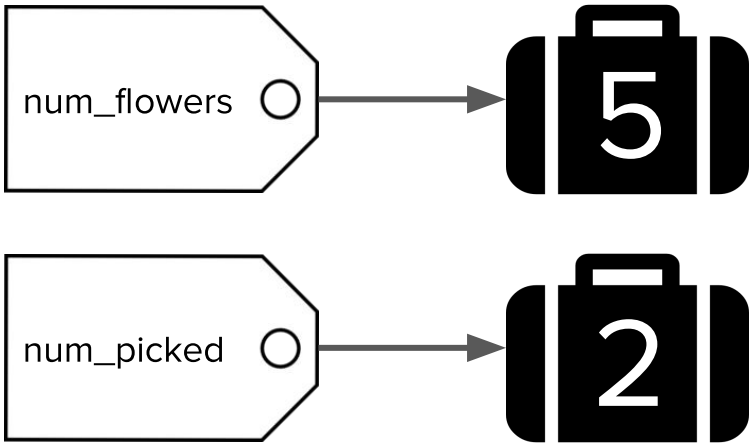
Think/Pair/Share: *Try to predict what happens here!*

An 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
```

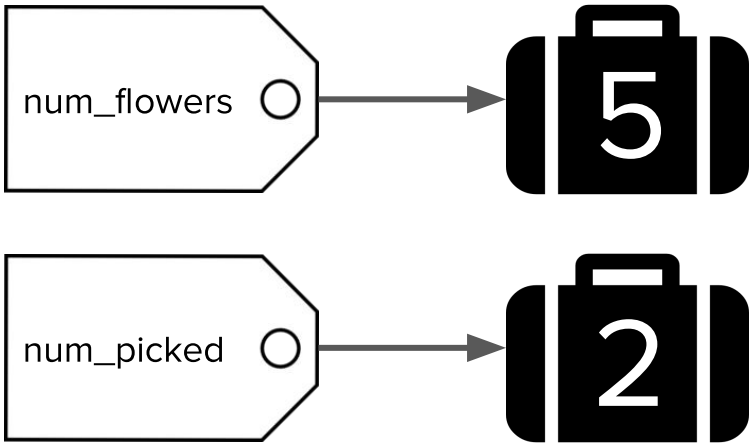
variable assignment!



An 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
```



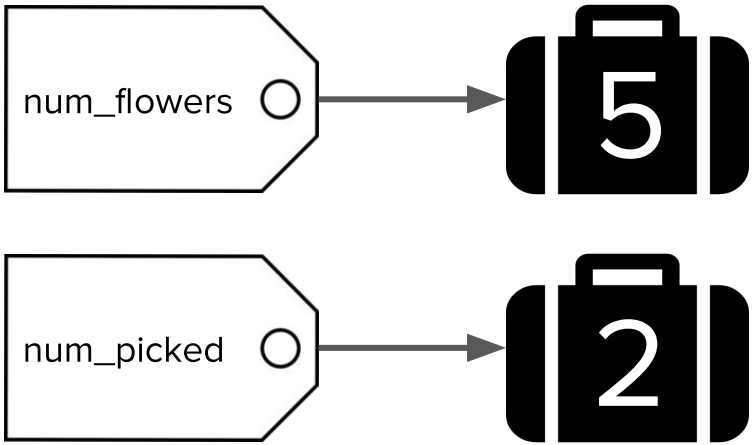
*The right side of the equals sign **always** gets evaluated first.*

An 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
```

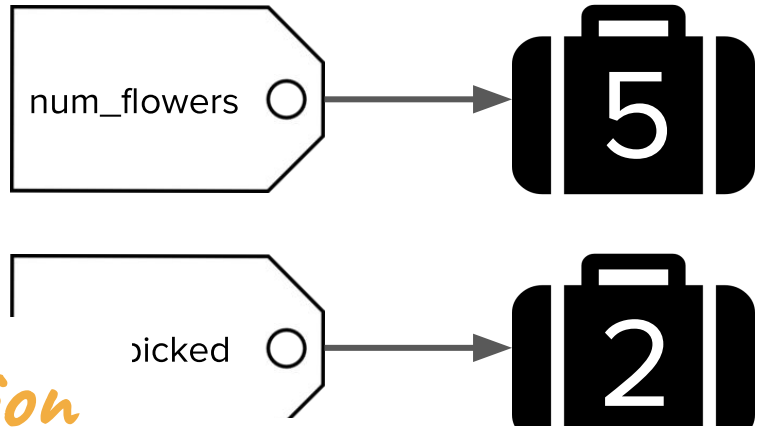
variable retrieval!



An 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

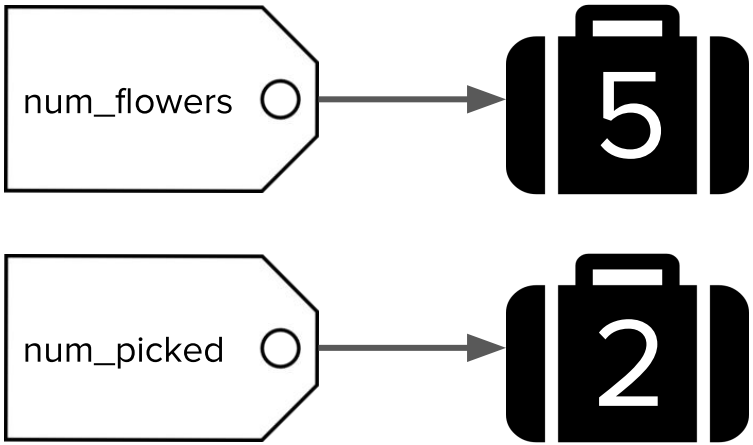
variable retrieval

The process of getting the value associated with a name

An 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
```



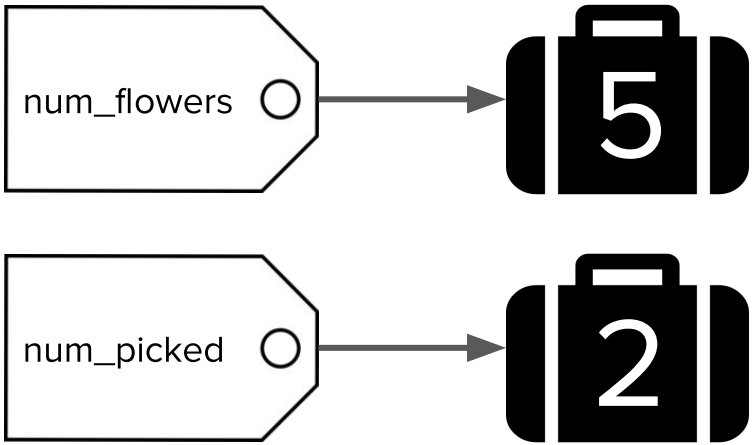
*The right side of the equals sign **always** gets evaluated first.*

An 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



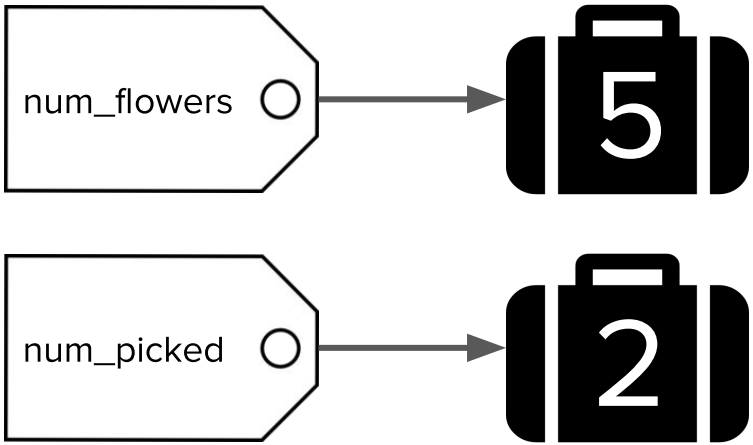
We get the values using variable retrieval (i.e. checking what suitcase is attached).

An 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



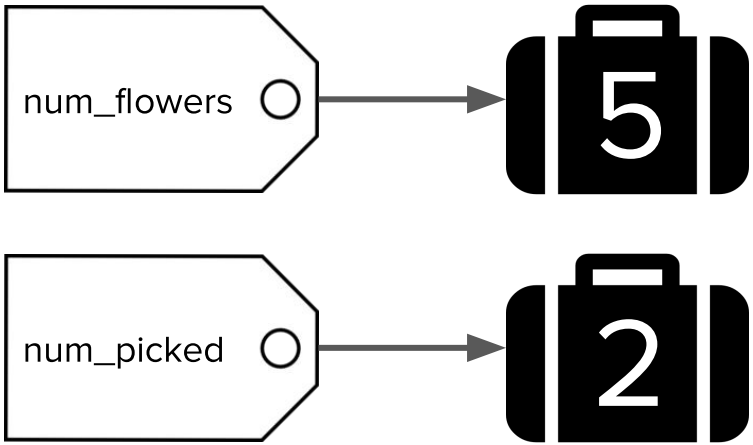
Then we can evaluate the right hand side of the assignment.

An 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



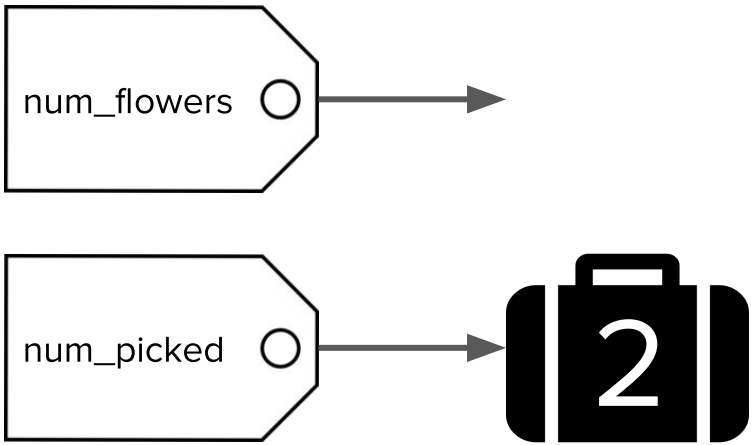
Then we can evaluate the right hand side of the assignment.

An 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



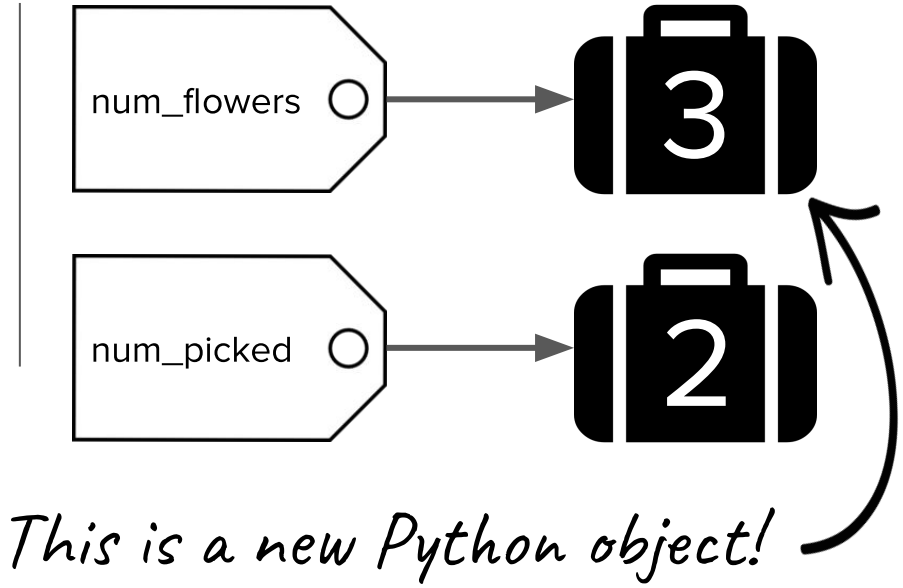
*The right side of the equals sign **always** gets evaluated first.*

An 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

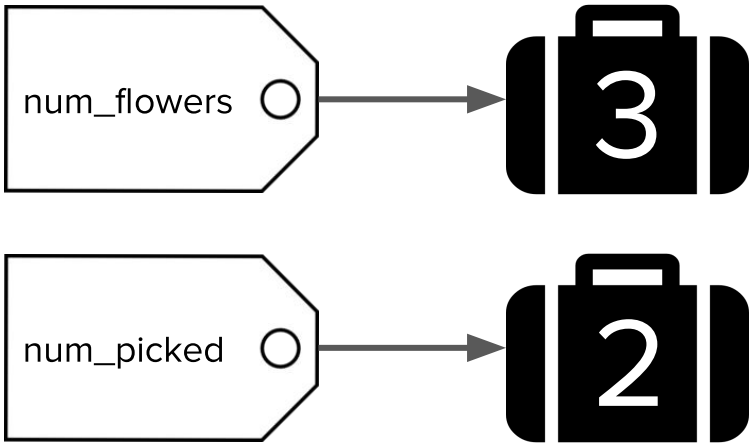


An example

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.

Terminology summary

- Variables have a **name** and are associated with a **value**

Terminology summary

- Variables have a **name** and are associated with a **value**

Style note

variable names
Have descriptive variable names that are nouns and that use snake_case.

Terminology summary

- Variables have a **name** and are associated with a **value**

- Variable **assignment** is the process of associating a value with the name (use the equals sign =)
  - This is how you create and update variables!

Terminology summary

- Variables have a **name** and are associated with a **value**
- Variable **assignment** is the process of associating a value with the name (use the equals sign =)
- **Retrieval** is the process of getting the value associated with the name (use the variable's name)
  - This is how you use variables!

Terminology summary

- Variables have a **name** and are associated with a **value**
- Variable **assignment** is the process of associating a value with the name (use the equals sign =)
- **Retrieval** is the process of getting the value associated with the name (use the variable's name)

The Python Interpreter

[demo]

What is the Python Interpreter?

- Comes with Python when you install it!

- Reads and executes Python code, and allows you to test single lines of code at a time.

- To use the built-in Python interpreter, open:
  - (Mac/Ubuntu) the Terminal application and type in “python3”
  - (Windows) the Command Prompt and type in “py”

- PyCharm also has a Python interpreter (click on the Python Console tab)!

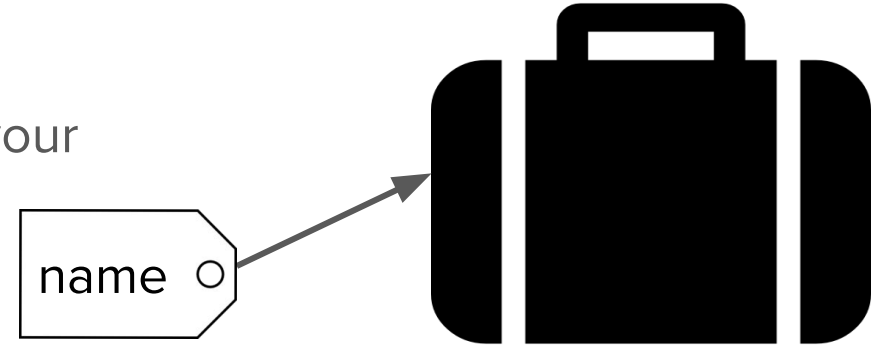
Types

The 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!

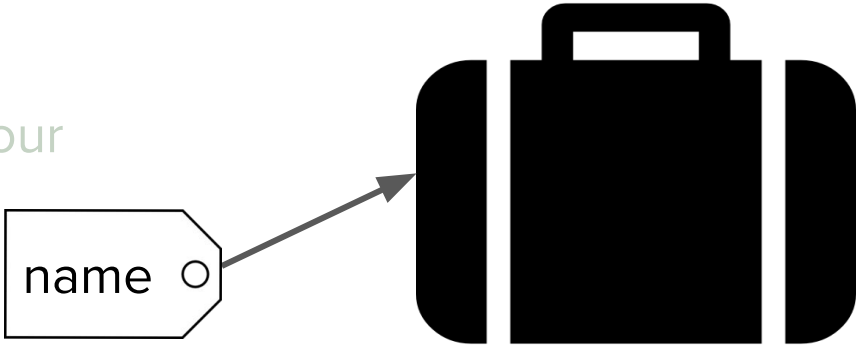


The suitcase analogy

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

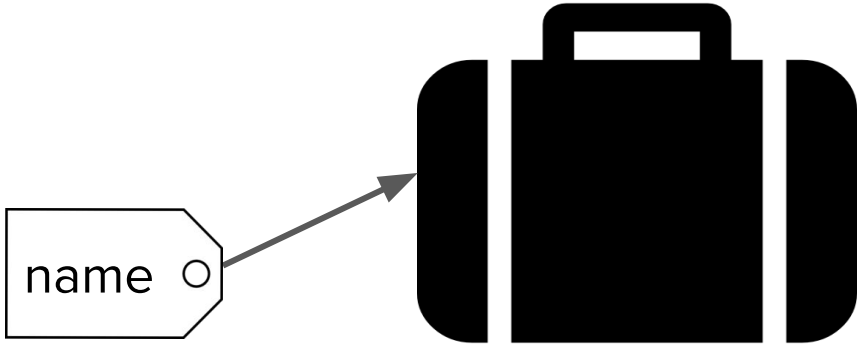
- You can think about a Python object as a suitcase stored in your computer's memory.

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



The suitcase analogy

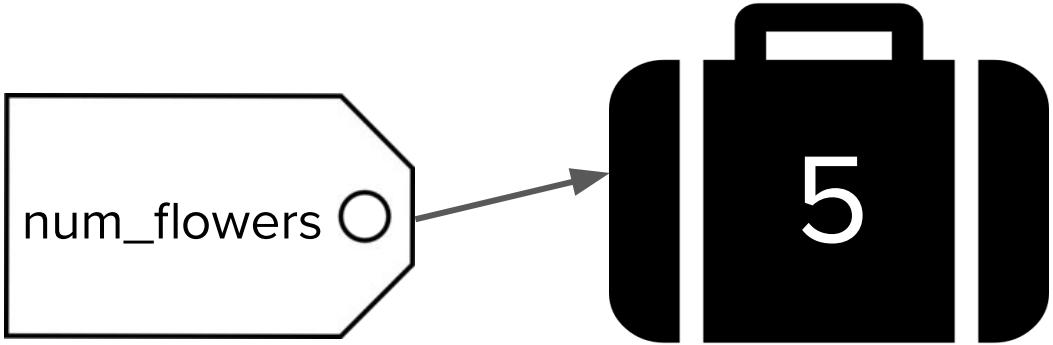
- When you store information in Python, it becomes a Python **object**
  - **Objects come in different sizes and types**



The suitcase analogy

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

```
num_flowers = 5
```



The suitcase analogy

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

```
num_flowers = 5
```

The suitcase analogy

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

```
num_flowers = 5
```

The suitcase analogy

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

```
num_flowers = 5.0
```

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

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
```

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
```

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

Called "doubles" in some other languages

```
fraction = 0.2
```

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
```

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

- We’ll learn more types later in the quarter!

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

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

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

Talk to your neighbor!

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

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

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

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

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**

`type(x)` can help us determine an object's type!

[demo]

Variables summary

- Information is stored in Python objects, which come in different types and sizes.
- Variables are like tags that associate names with Python objects.
- We can use `type(x)` to figure out the type of the Python object the variable holds.
- Sometimes we **initialize** variables to `None` before assigning them another value.

How do we use 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`

Arithmetic operators

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

Arithmetic operators

*	Multiplication
/	Division
//	Integer division (**takes the largest integer smaller than the answer**)
%	Modulus (remainder)
+	Addition
-	Subtraction

Arithmetic operators

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

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

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

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

Talk with a partner!

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]

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]

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

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!

Boolean comparison operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Boolean comparison operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

The resulting expression always evaluates to true or false (but the operators don't need to compare two booleans).

Boolean comparison operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Therefore, we can use the resulting expression as the condition in if statements and while loops!

Boolean comparison operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Operator	Precedence
`()`	1
`*, /, //, %`	2
`+, -`	3
`<, <=, >, >=, ==, !=`	4

Boolean comparison operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Operator	Precedence
`()`	1
`*, /, //, %`	2
`+, -`	3
`<, <=, >, >=, ==, !=`	4

Ties still broken by going left to right!

Boolean comparison operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Operator	Precedence
`()`	1
`*, /, //, %`	2
`+, -`	3
`<, <=, >, >=, ==, !=`	4

[demo]

What if the operator isn't built in?

The `math` library

- A library of non-built-in math functions for Python

The **`math`** library

- A library of non-built-in math functions for Python

- We have to use a “noun.verb()” format to call the functions in the library.
  - For example: **`math.sqrt(4)`**

The **`math`** library

- A library of non-built-in math functions for Python

- We have to use a “noun.verb()” format to call the functions in the library.
  - For example: **`math.sqrt(4)`**

Name of library

Library function

The `math` library

- A library of non-built-in math functions for Python

- We have to use a “noun.verb()” format to call the functions in the library.
  - For example: `math.sqrt(4)`

- We have to `import math` at the top of our program file to use the library
  - We’ll do this for you in Assignment 2!

The **`math`** library

- A library of non-built-in math functions for Python

- We have to use a “noun.verb()” format to call the functions in the library.
  - For example: **`math.sqrt(4)`**

- We have to **`import math`** at the top of our program file to use the library

[demo]

Expressions summary

- The computer **evaluates** expressions to a single value.
  - In Python, expressions can evaluate to any type!

Expressions summary

- The computer **evaluates** expressions to a single value.

- We use **operators** to combine literals and/or variables into **expressions**.
  - We have both arithmetic and comparison operators.
  - Operators have a certain **precedence**.

Expressions summary

- The computer **evaluates** expressions to a single value.

- We use **operators** to combine literals and/or variables into **expressions**.

- If an expression results in a true or false value, it can be used as the **condition** in control flow structures we've already learned!
  - I.e. if statements and while loops

Expressions summary

- The computer **evaluates** expressions to a single value.
- We use **operators** to combine literals and/or variables into **expressions**.
- If an expression results in a true or false value, it can be used as the **condition** in control flow structures we've already learned!
- The **`math`** library provides functions to help with non-built-in arithmetic operators.
  - E.g. **`math.sqrt(value)`**

Expressions summary

- The computer **evaluates** expressions to a single value.
- We use **operators** to combine literals and/or variables into **expressions**.
- If an expression results in a true or false value, it can be used as the **condition** in control flow structures we've already learned!
- The **`math`** library provides functions to help with non-built-in arithmetic operators.

Variables and functions

We can display variables' values using **`print()`**

```
def print_my_variables():

    x = 5

    y = 2

    print(x)

    print(y)

    print(x + y)
```

We can display variables' values using **`print()`**

```
def print_my_variables():
    x = 5
    y = 2
    print(x)
    print(y)
    print(x + y)
```

[demo]

We can display variables' values using **`print()`**

```
def print_my_variables():
    x = 5
    y = 2
    print(x)
    print(y)
    print(x + y)
```

NOTE: Variables only exist in the function in which they're created

We can display variables' values using **`print()`**

```
def print_my_variables():

    x = 5

    y = 2

    print(x)

    print(y)

    print(x + y)
```

`print()` *displays the value of what's inside the parentheses*

We can display variables' values using **`print()`**

```
def print_my_variables():

    x = 5

    y = 2

    print(x)

    print(y)

    print(x + y)
```

In functions, we can't just type **`x`** *or* **`y`** *directly like we did in the interpreter.*

We can display variables' values using **`print()`**

```
def print_my_variables():

    x = 5

    y = 2

    print(x)

    print(y)

    print(x + y)
```

We'll go into more depth about **`print()`** *later this week!*

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 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 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 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)

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)

Let’s put it all together!

Receipt program

- We'll create a variable for the total_cost
- Tax: 7.25 percent of the total_cost
- Tip:
  - If num_people > 5: 20 percent of total_cost
  - Else: 15 percent of total_cost

Receipt program

- We'll create a variable for the total_cost
- Tax: **7.25** percent of the total_cost
- Tip:
  - If num_people > **5**: **20** percent of total_cost
  - Else: **15** percent of total_cost

Constants!

Receipt program

- We'll create a variable for the total_cost
- Tax: 7.25 percent of the total_cost
- Tip:
  - If num_people > 5: 20 percent of total_cost
  - Else: 15 percent of total_cost

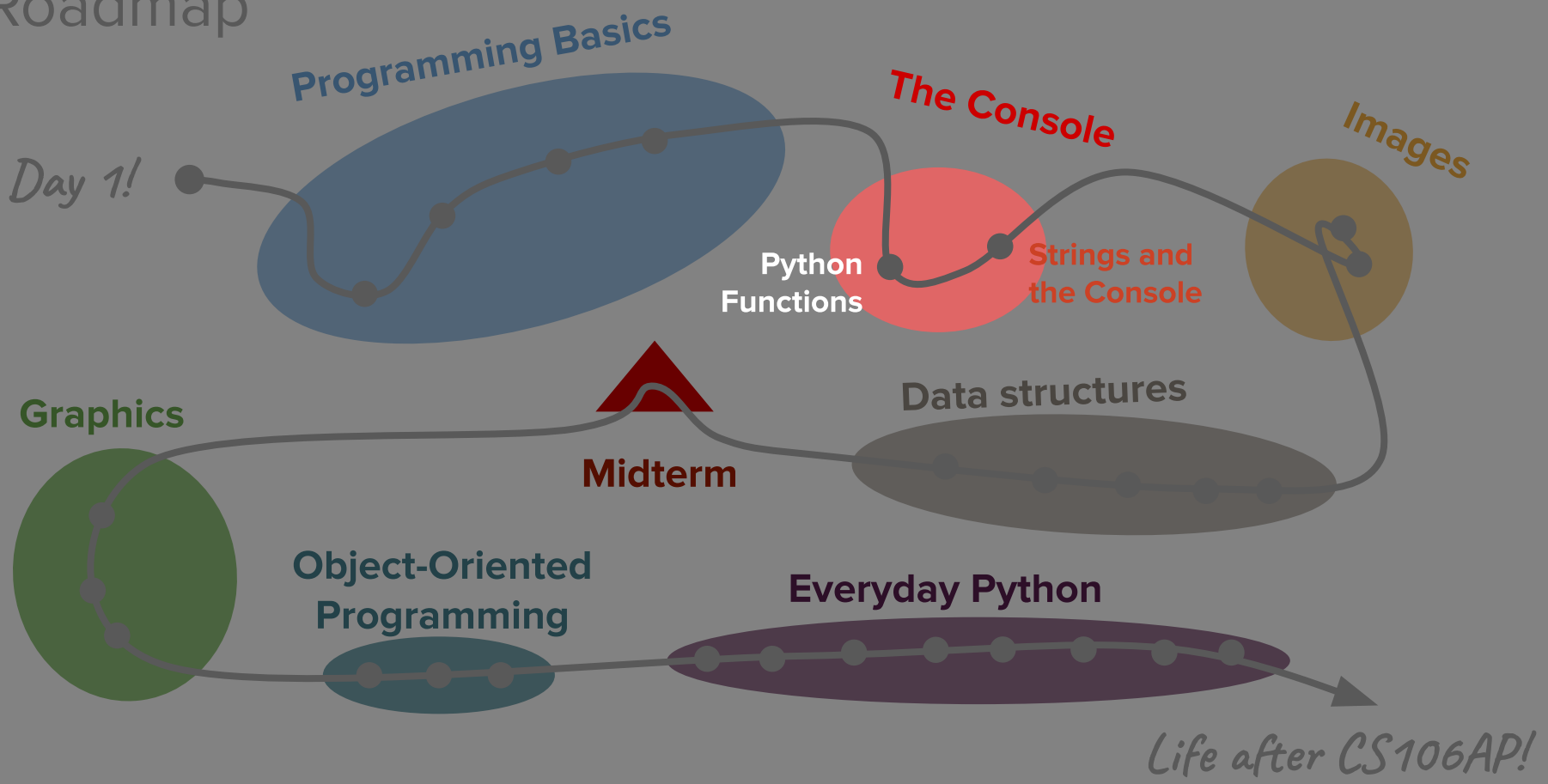
[demo]

What’s next?

This week: A better receipt program

- Can we decompose our program?
- What if the total cost changes and we want to calculate the receipt again?
- What if we want other people to be able to use our program?

Roadmap


Programming Basics
Day 1!
The Console
Images
Python Functions
Strings and the Console
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!