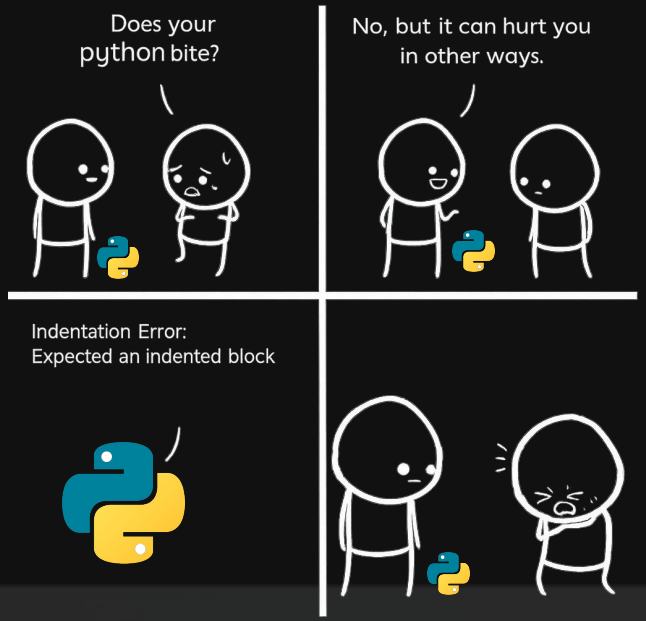
HTGAA Bio-Bootcamp Part III



Python Basics

Jan 23, 2026

Paul Kao

Image Source: x.com. @okaypriyanka

mit media lab

Python is the most popular programming languages in 2025

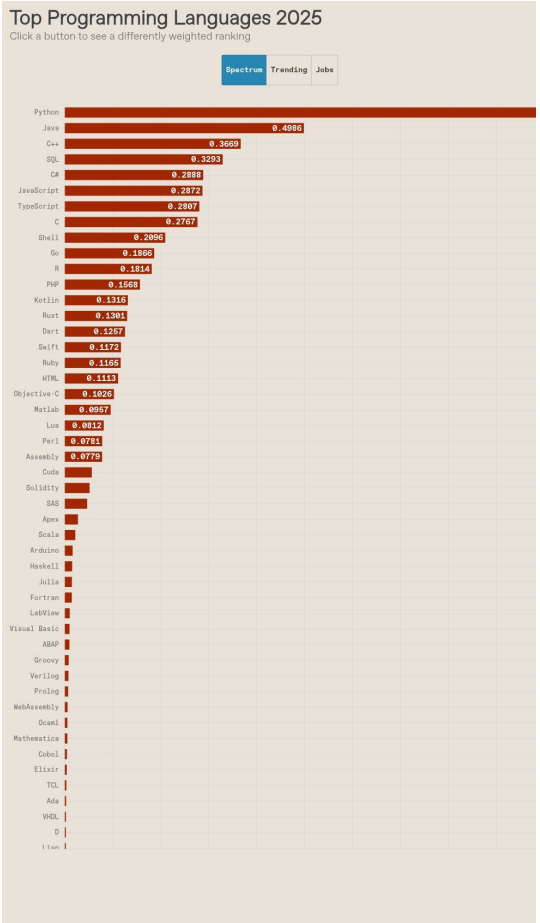


Image Source: IEEE Spectrum, The Top Programming Languages 2025

Scientific Python Ecosystem

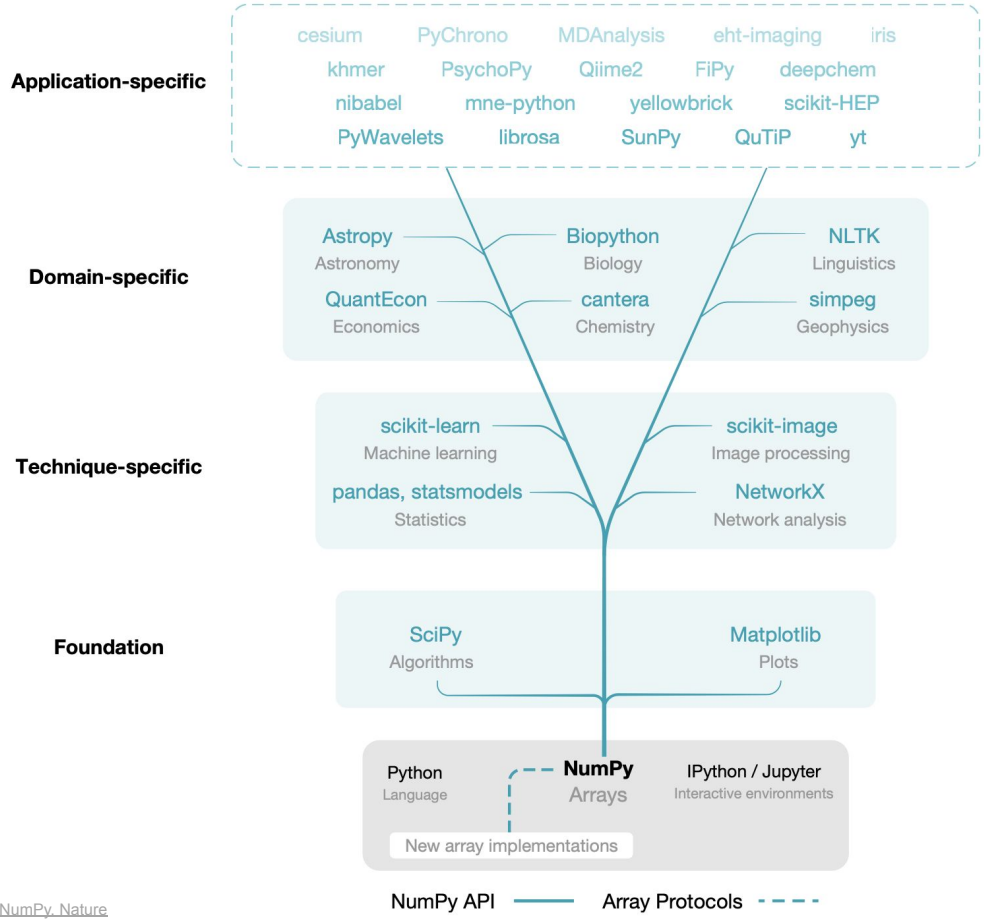


Image Source: Harris et al (2020). Array programming with NumPy. Nature

Benefits of Learning Python

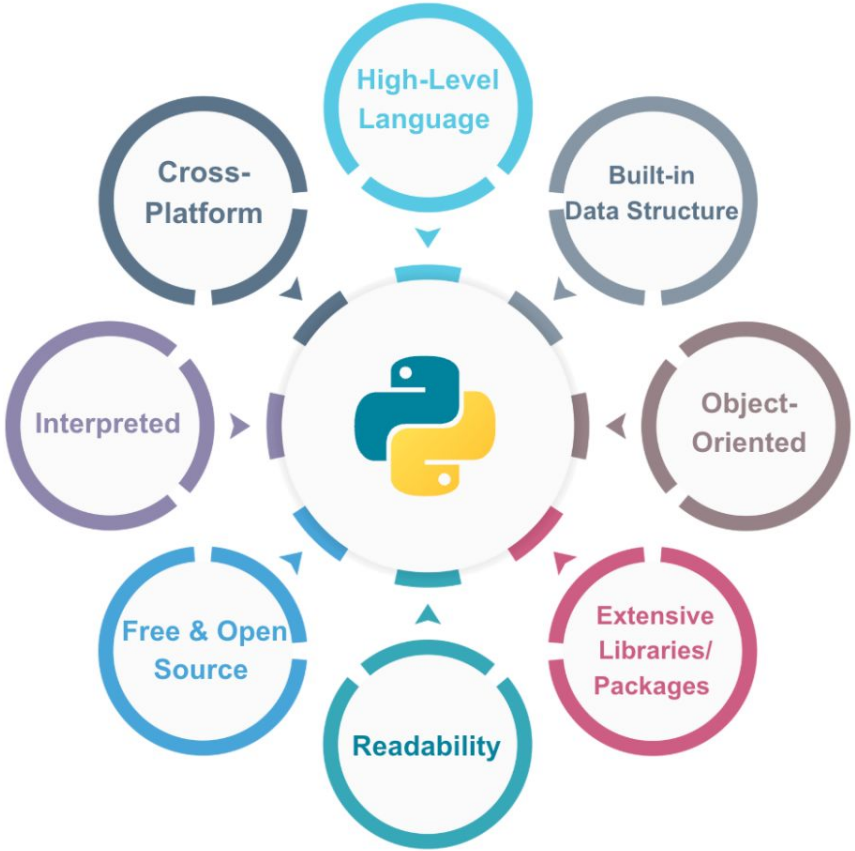
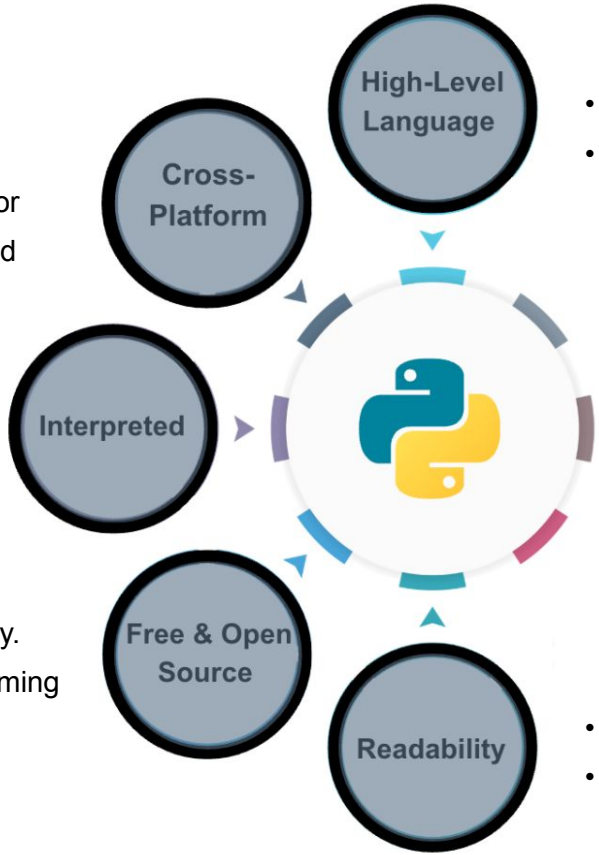


Image Source: www.javatpoint.com

Benefits of Learning Python

- Execute the code line by line. Any error happens, it stops further execution and reports back. ***More Details***



- Abstract low-level details for ease of use.
- Require fewer lines of code than languages like C/C++.

- Free to use and large active community.
- Python, R, Julia (widely-used programming languages in Bioinformatics) are all open-sourced.

- English-like syntax. Easy to Read, Learn and Write.
- Coding Style. Follow guides like ***PEP 8***, ***PEP 20***.

Image Source: www.javatpoint.com

Today, We Will Cover the Following Topics:

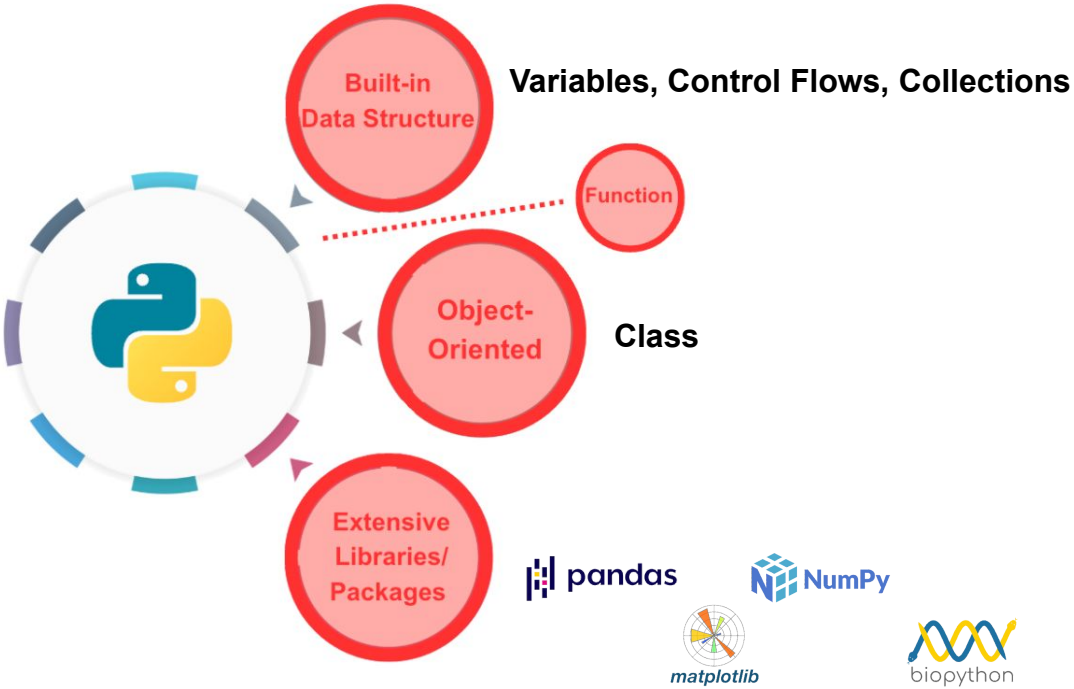
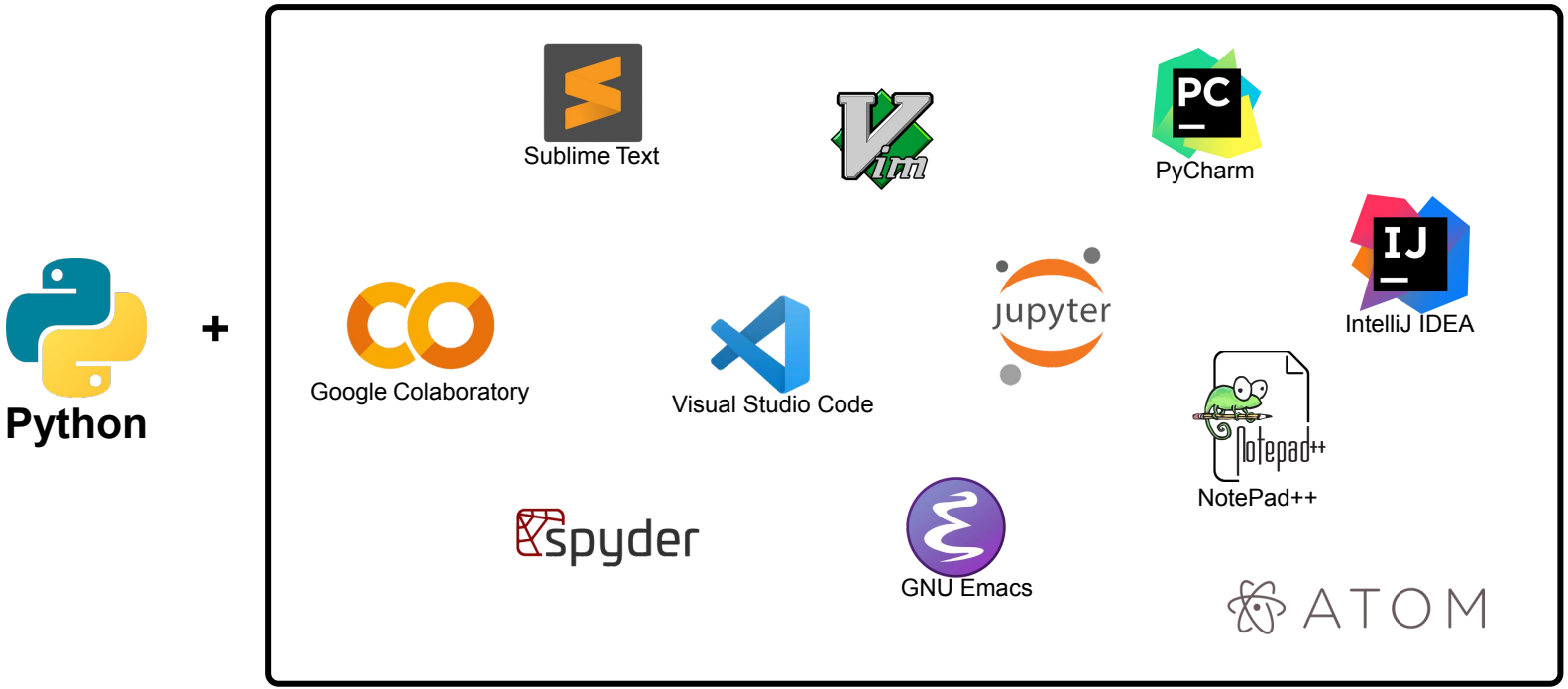


Image Source: www.javatpoint.com

To Run a Python Code, You Need ...



We Choose to Use Google Colab

Benefits of Using Google Colab

Cloud-Based & No need for complex configuration setup

- Run Python in your browser effortlessly.

Pre-Installed Libraries

- Access a wide range of Python libraries out-of-the-box.

Integration with Google Drive

- Save, manage and access your work seamlessly through Google Drive.

Free Access to GPUs

- Enhance performance for machine learning and other data-heavy operations.

Interactive Environment

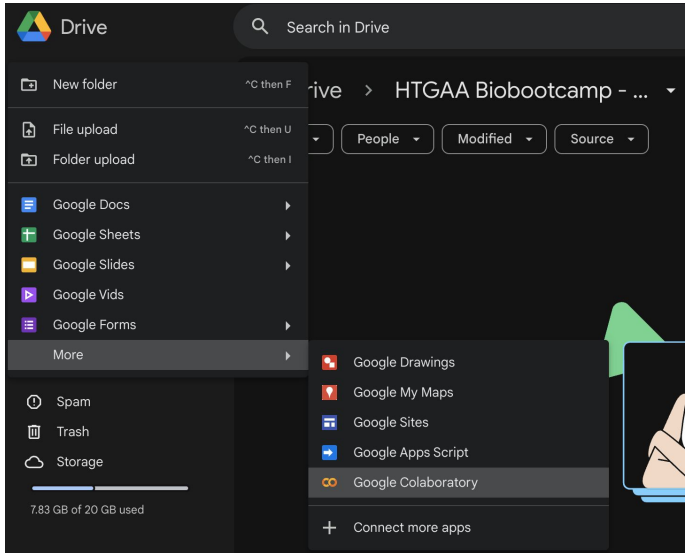
- Supports rich media, charts and interactive widgets.

Source: Google Colab: Cloud based Python IDE

Google Colab Setup

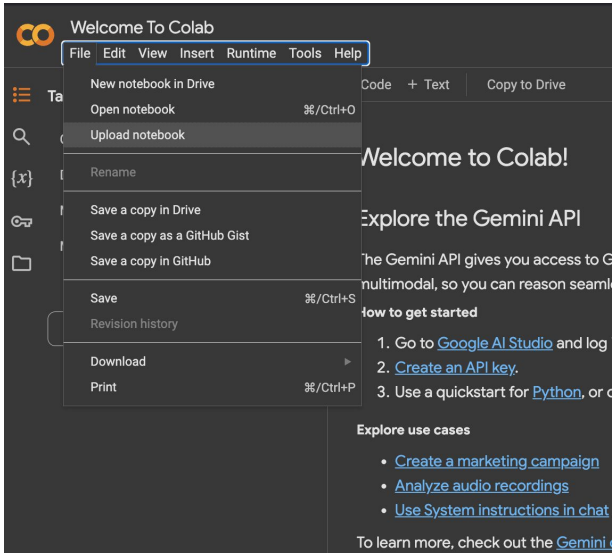
Option 1: From Google Drive

1. **Log in to your Google Drive**
2. **Click New → More → Google Colaboratory**
   (If you don't see it, install from the *Marketplace*.)



Option 2: From Colab Website

1. **Go to *Google Colab***
2. **Upload an `.ipynb` File**
   (Select File → Upload notebook)
   - ***Today's Colab Example***



Expression vs. Statements

Expression: A piece of code that produces a value

operands

Ex: `3 + 4` , `"Hello" + "World"` , `2 * 5 + 1`

operator

Some operators are applied before others

*, / and % happen before + and -

Statement: A piece of code that performs an action → it tells the program what to do

Ex:

1.
```
x = 3 + 4          # assignment statement
```

2.
```
if x > 5:          # if statement
    ...
```

3.
```
while x < 10:      # while statement
    ...
```

4.
```
for i in range(5): # Loop (for)
    ...
```

Source: geeksforgeeks, Expressions in Python

Source: 1.1 Programming Model. Algorithms, 4th Edition. Robert Sedgewick and Kevin Wayne

Demo

Assignment (0/6)

line that just executed
next line to execute

Left Right

```
1  a = "Peter"
2  b = 16
3
4  c = 15 + 17
5  d = b + 3
6
7  c = c - b
```

Visualize the Process of Execution:

- Use ***Python Tutor*** to see how each line of code is executed in real time.
  - Click on the *Edit Code & Get AI Help*.

Be the Green Arrow :

- Imagine yourself as the green arrow and move line by line.

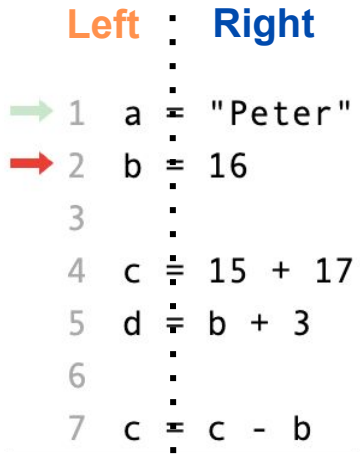
Focus on the **Left** = **Right** in each line.

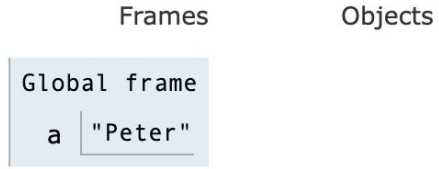
Demo

Source: CS61A, UC Berkeley, Prof. John DeNero

Assignment (1/6)

line that just executed
next line to execute



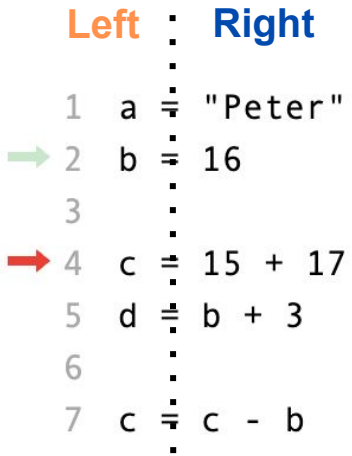


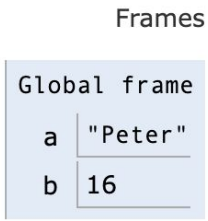
1 **An expression called** `"Peter"` **(String Value)** **(= Right) is evaluated, and its value is bound to the variable named** `a` **(Left =)**

Source: CS61A, UC Berkeley, Prof. John DeNero

Assignment (2/6)

line that just executed
next line to execute



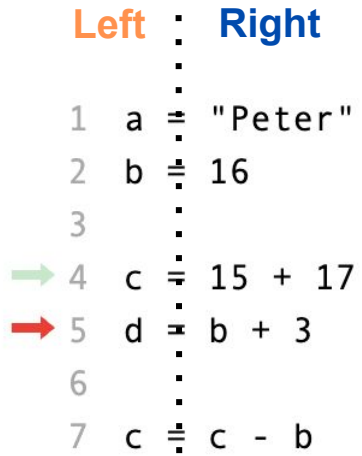


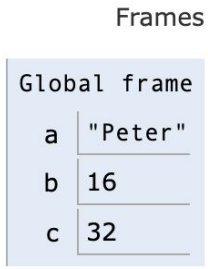
2 **An expression called (Integer Value)** 16 **(= Right) is evaluated, and its value is bound to the variable named** b **(Left =)**

Source: CS61A, UC Berkeley, Prof. John DeNero

Assignment (3/6)

line that just executed
next line to execute





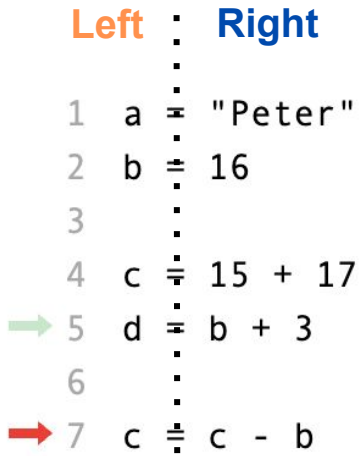
4 **An expression called** `15 + 17` **(= Right) is evaluated, and its value is bound to the variable named** `c` **(Left =)**

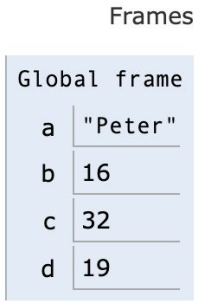
which will bring us to 32

Source: CS61A, UC Berkeley, Prof. John DeNero

Assignment (4/6)

line that just executed
next line to execute





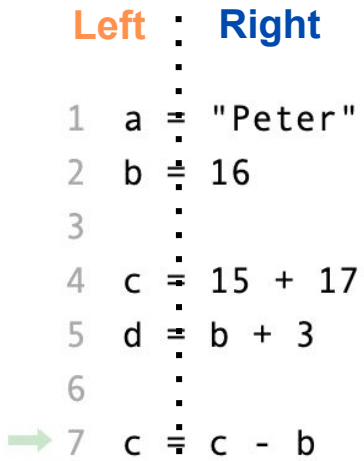
5 **An expression called** `b + 3` **(= Right) is evaluated, and its value is bound to the variable named** `d` **(Left =)**

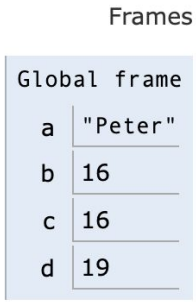
16 + 3, which will bring us to 19

Source: CS61A, UC Berkeley, Prof. John DeNero

Assignment (5/6)

line that just executed
next line to execute





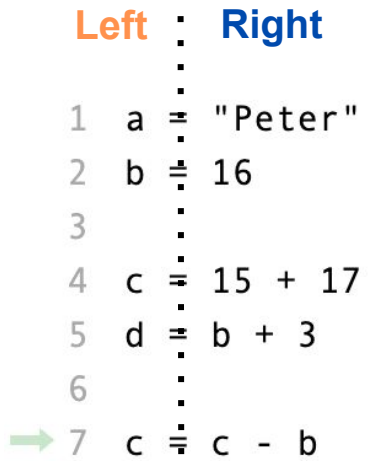
7 **An expression called** `c - b` **(= Right) is evaluated, and its value is bound to the variable named** `c` **(Left =)**

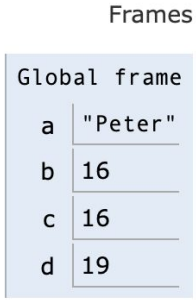
32 - 16, which will bring us to 16

Source: CS61A, UC Berkeley, Prof. John DeNero

Assignment (6/6)

line that just executed
next line to execute





7 **An expression called** `c - b` **(= Right) is evaluated, and its value is bound to the variable named** `c` **(Left =)**

32 - 16, which will bring us to 16

Execution Rule for Assignment Statements:

1. **All Expressions on the Right side of = are computed from Left to Right.**
2. **Assign result(s)** from 1. **to the Variable(s) on the Left side of = in the current frame.**

Source: CS61A, UC Berkeley, Prof. John DeNero

Control Flow

- Decide which parts of your code run (and how many times) based on conditions.
  They help you organize logic and promote code reusability.

```
if condition:
    # Execute when condition is True
else:
    # Execute otherwise
```

```
while condition:
    # Keep executing as long as condition is True
```

```
for val in sequence:
    # Repeat for each item in sequence
```

Collections

- Built-in data structures that store multiple items in a single variable.
  Each types have distinct rules for ordering, mutability, and duplication.

Type	Allows Duplicates?	Mutability (can it be changed after being created?)	Ordering	Brackets
List	Yes	Changeable (mutable)	Ordered	[ ]
Tuple	Yes	Not changeable (immutable)	Ordered	( )
Dictionary	No (for keys)	Changeable (mutable), indexed by keys	Unordered	**items** { } (**key:value** pairs)
Set	No	Cannot be changed in place, but can add items; non-indexed	Unordered	{ }

Table Source: Medium. Python Collections — List, Tuple, Dictionaries & Sets

Demo

Function

- A function is a reusable block of code that runs only when called.
  It helps keep your program organized and easier to maintain.

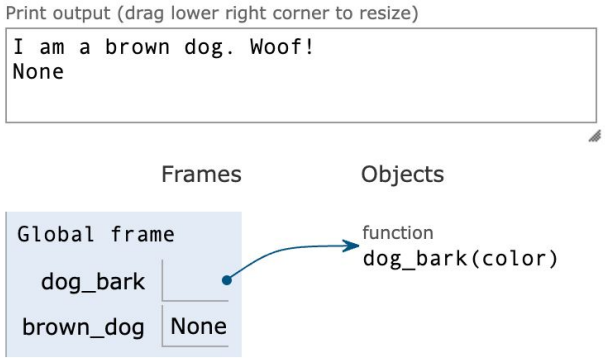
Variables / Arguments

```
def function_name(parameters):
    # statements
    return expression # Optional; returns a value
```

Ends function call & Send data back to the program

```
def dog_bark(color):
    """
    Prints a message about the dog's color.
    Returns:
        None
    """
    print(f"I am a {color} dog. Woof!")
    # No return statement

brown_dog = dog_bark("brown")
print(brown_dog) # Output: None
```



Class

- A class is a blueprint for creating objects.
- You can create your own class to define attributes (***with variables***) and methods / behaviors (***with functions***).

```python
class Dog:
    def __init__(self, color, name):
        self.color = color      # attribute
        self.name  = name       # attribute

    def dog_bark(self):         # method / behavior
        return f"I am a {self.color} dog named {self.name}. Woof!"
```

Dog (class)

- **color** *attribute*
- **name** *attribute*

dog_bark() *method*

- **self** refers to the current instance (object).
- a **dot(.)** in **object.attribute** or **object.method()** means you are accessing an attribute or invoking a method on that particular object (or class).
- **__init__:** a special method called automatically when creating a new Dog object.

starts and ends with two underscores

Class

- A class is a blueprint for creating objects.
- You can create your own class to define attributes (***with variables***) and methods / behaviors (***with functions***).

```python
class Dog:
    def __init__(self, color, name):
        self.color = color      # attribute
        self.name  = name       # attribute

    def dog_bark(self):         # method / behavior
        return f"I am a {self.color} dog named {self.name}. Woof!"

# Create an instance (object) of the Dog class
chill_dog = Dog("brown", "chillguy")
# Create an instance (object) of the Dog class
nochill_dog = Dog("yellow", "no_chillguy")

print(chill_dog.dog_bark())
# Output: I am a brown dog named chillguy. Woof!"
```

Dog (class)

- **color** *attribute*
- **name** *attribute*
- **dog_bark()** *method*

chill_dog (instance object)

- **color:** *brown*
- **name:** *chillguy*
- **dog_bark()**

nochill_dog (instance object)

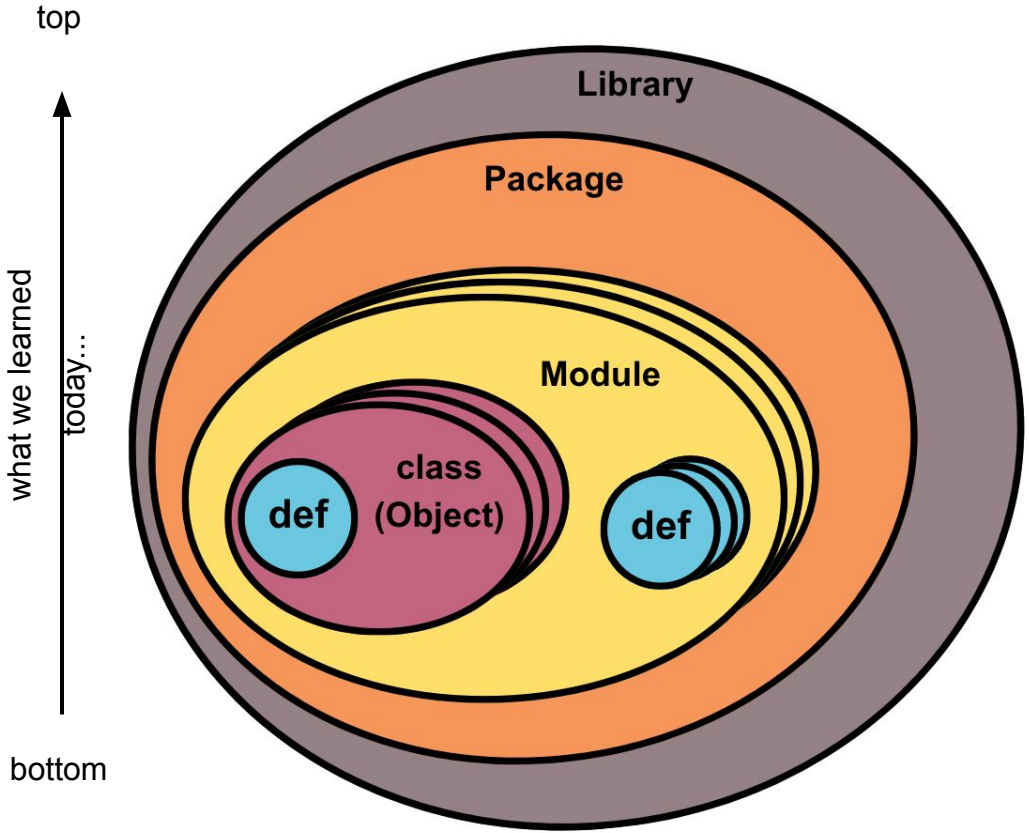
- **color:** *yellow*
- **name:** *no_chillguy*
- **dog_bark()**

More example about class (github) - Linear Regression

Image Source: ChillGuy, Philip Banks , KC Green, On Fire

Demo

When Building a Python Project...




```
pip install <package / library>
```

Library:

- a collection of related modules and packages.
- an umbrella term refers to a reusable chunk of code.

Ex: NumPy, matplotlib, PyTorch, biopython

"

Package: a collection of modules.

Ex: pandas

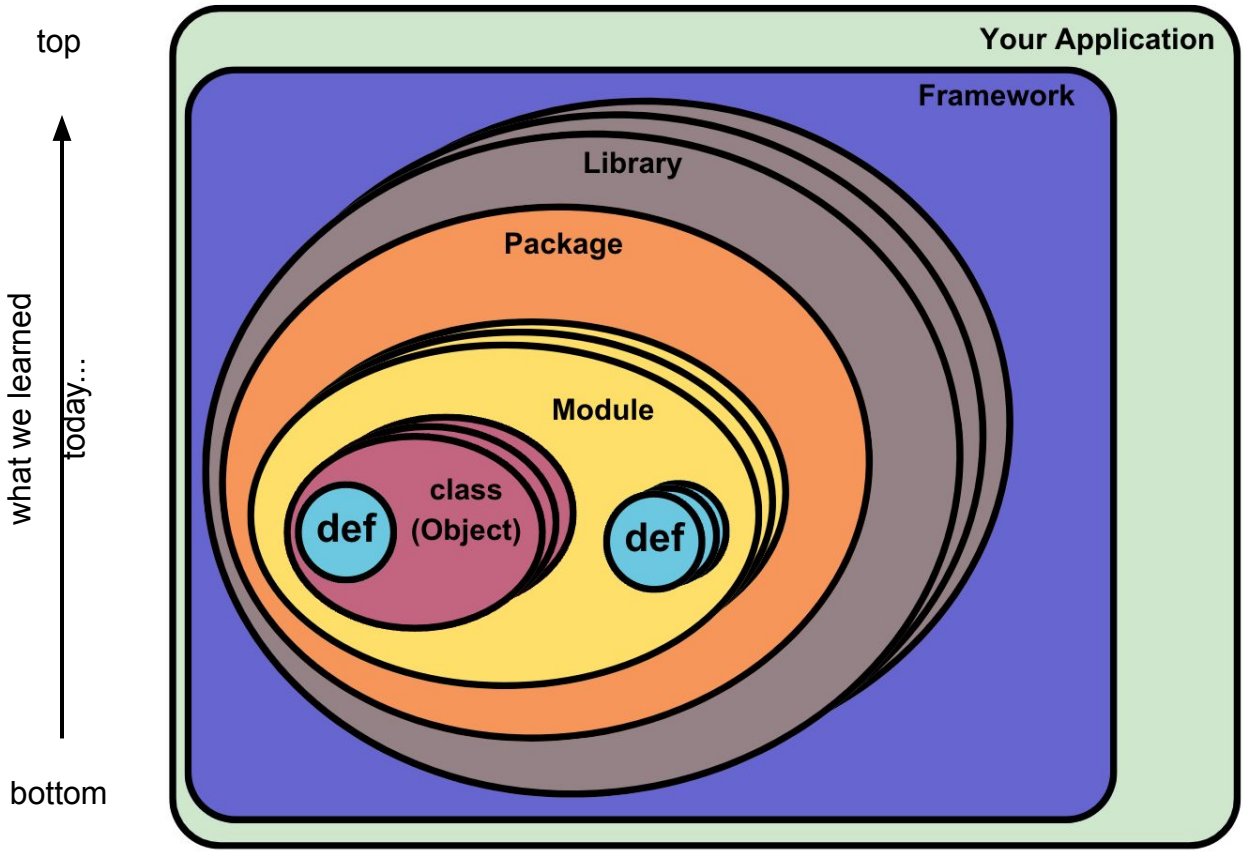
Module: a collection of functions or objects.

Ex: random, datetime

Source: Medium, 6 Must-know words in Python
Source: Medium, The Difference between Module, Package, Library, and Framework

Demo

When Building a Python Project...



Think TOP-DOWN: start with picking a framework, add libraries, then define custom code if needed

top

bottom

Source: Medium. 6 Must-know words in Python
Source: Medium. The Difference between Module, Package, Library, and Framework

Quiz

Q1. What happens when you run this code?

```
class A:
    x = 1

a1 = A()
a2 = A()
a1.x = 5
print(a1.x, a2.x, A.x)
```

A. 5 5 5
B. 5 1 1
C. 5 1 5
D. 1 1 1

Q2. What does PEP 20 (The Zen of Python) say? Select all that apply.
(Hint: Open a Python shell and run `import this`)

A. Readability counts.
B. Errors should pass easily.
C. Nested is better.
D. Namespaces are one honking great idea -- let's do more of those!

Q3. Open a Python shell in your terminal (check HTGAA Pre Bootcamp Python Course). Find the **smallest** integer n that satisfies **ALL** of these:

1. 2000 < n < 2500
2. n leaves remainder 6 when divided by 10 (`n % 10 == 6`)
3. n leaves remainder 3 when divided by 7 (`n % 7 == 3`)

Takeaways

Be the Green Arrow **and follow each line as it executes.**

Pay attention to the error messages. Fixing them sharpens your skills.

Learn from bottom to top, but build from top to bottom!

Keep Grinding! Divide & Conquer :D

Feel free to drop me a line If anything is unclear or incorrect!