Python: Day 01

Introduction to Python and Basic Syntax

Objectives

Foster a Strong Foundation

Understand the fundamental components and how to use them correctly

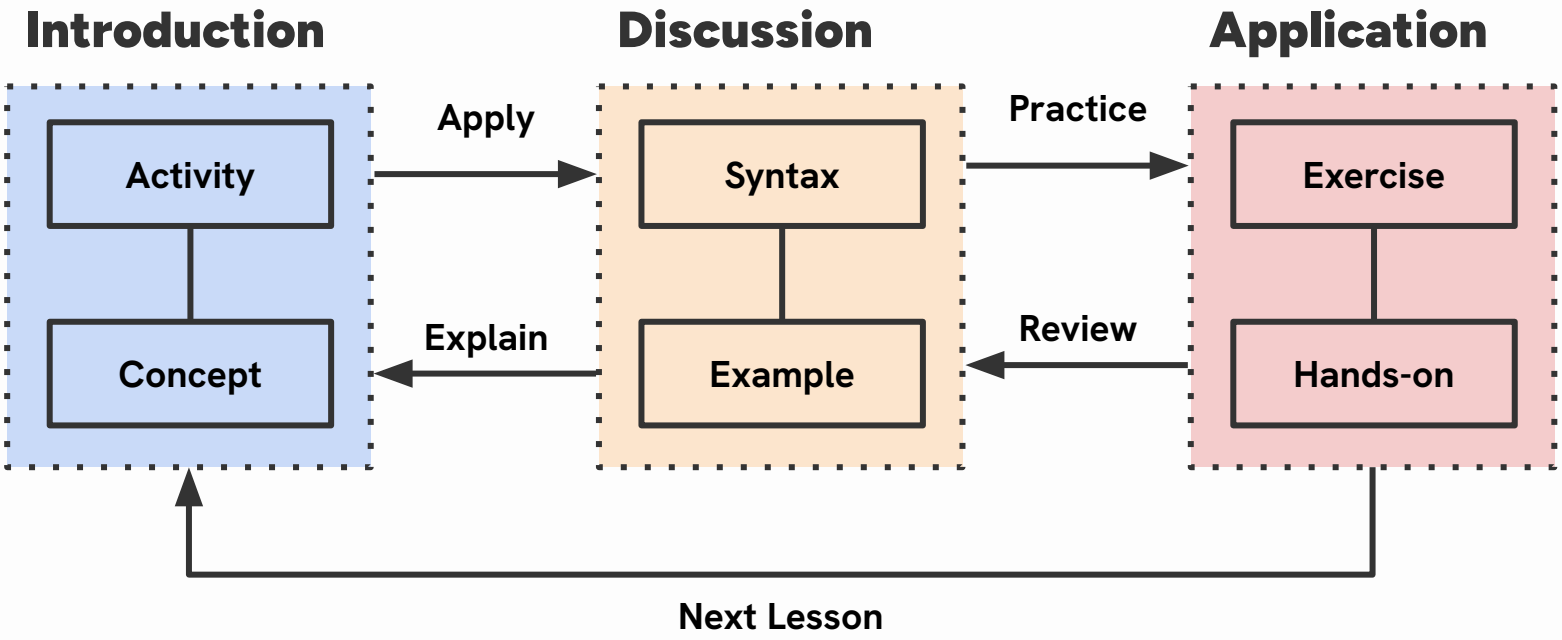
Develop Problem Solving Skills

Gain practical experience through activities, exercises, and lab sessions

Prepare for Specialization

Provide a preview on how Python can be used in various industries

Structure


Introduction
Discussion
Application
Activity
Concept
Apply
Explain
Syntax
Example
Practice
Review
Exercise
Hands-on
Next Lesson

Agenda

01

Introduction

What is Python?

02

Variables

Data Storage

03

Control Flow

Processing Information

04

Functions

Grouping Control Flows

05

Error Handling

Handling invalid code

06

Lab Session

Culminating Exercise

01

Introduction

Overview of Python’s characteristics and potential

Key Features

Convenient

Simple and concise
for easier development

Modern

Constantly updated with
useful features

Active

Large community with a
rich ecosystem

"Python is a clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java. "

— python.org

Java

```java
class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello, World");
    }
}
```

C++

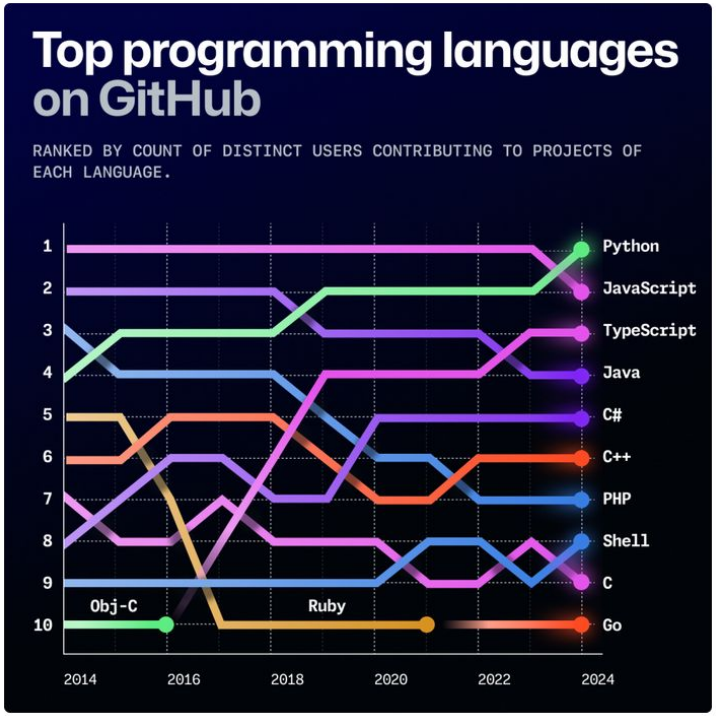
```cpp
#include <iostream>

int main() {
    std::cout << "Hello World" << std::endl;
}
```

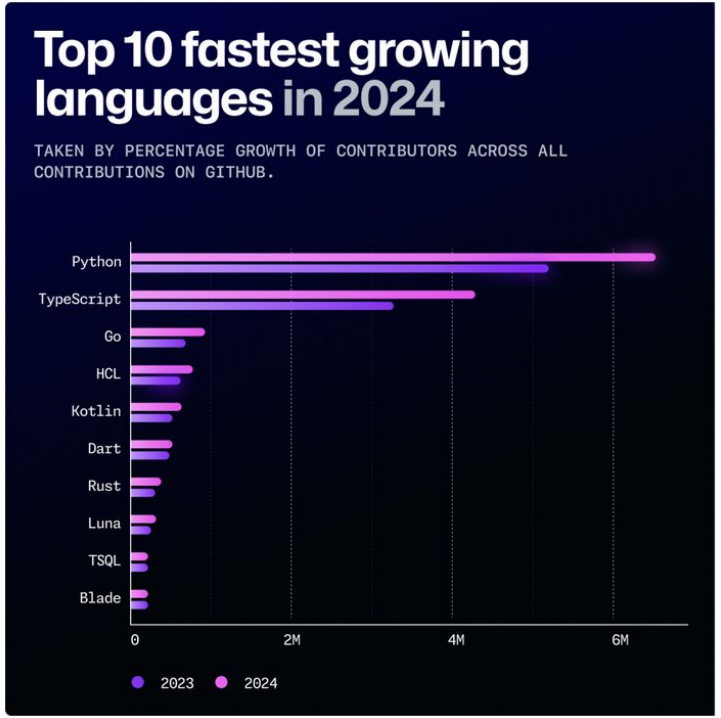
Python

```
1 print("Hello World")
```

Python Package Index (pypi.org)

Python Growth





Where can you use Python?

Data Science

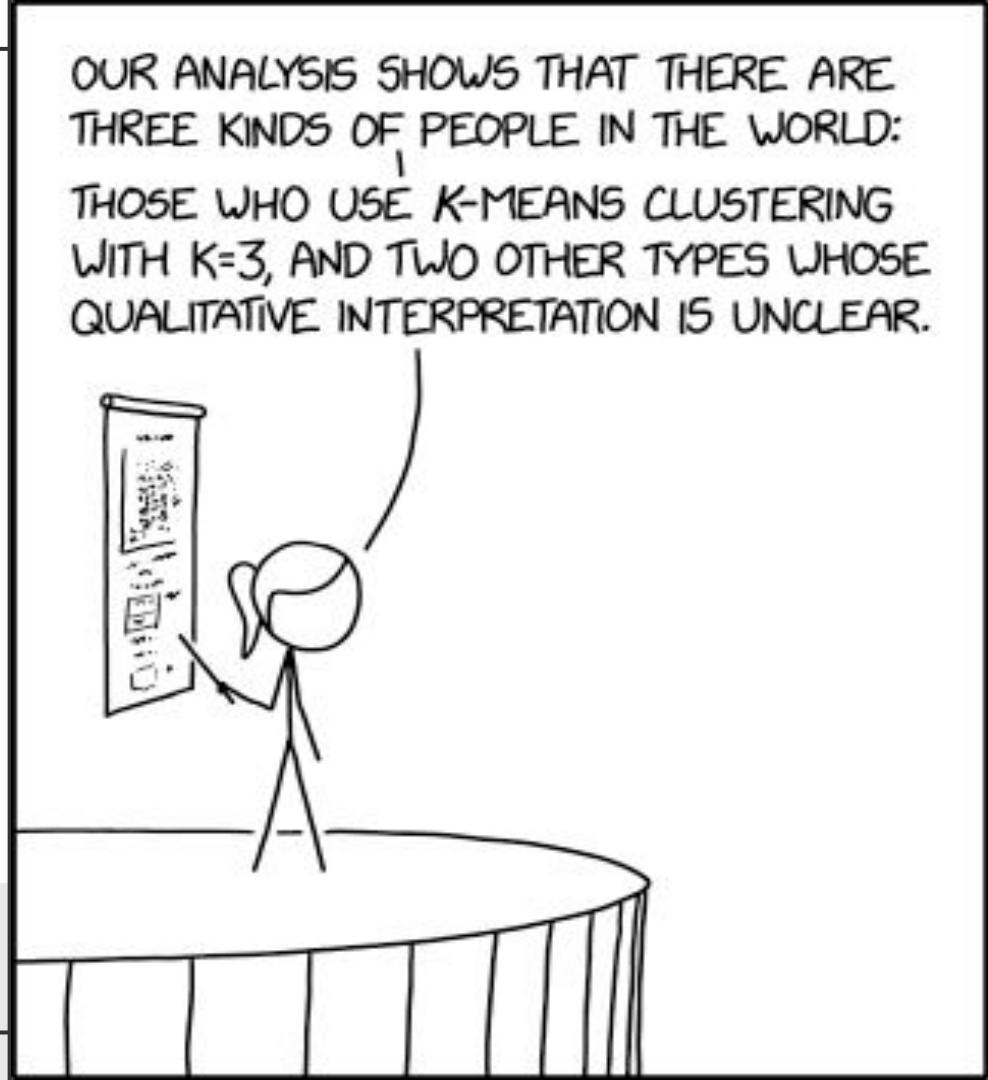
Python is famous for data science libraries specializing in data cleaning, visualization, and modelling.

pandas

scikit learn

seaborn

plotly



Machine Learning

Math-intensive processes in machine learning are often made in low-level languages and interfaced with Python

PyTorch

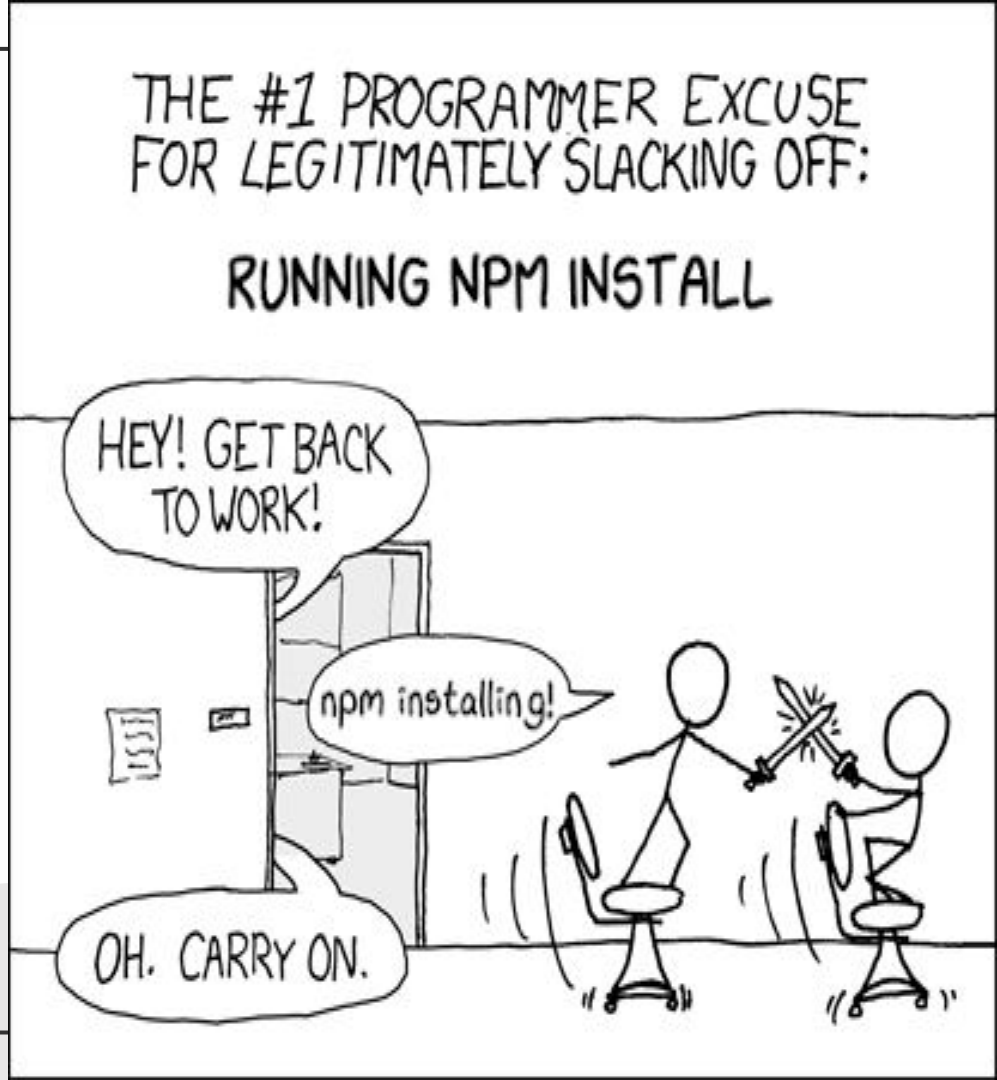
TensorFlow

spaCy

OpenCV

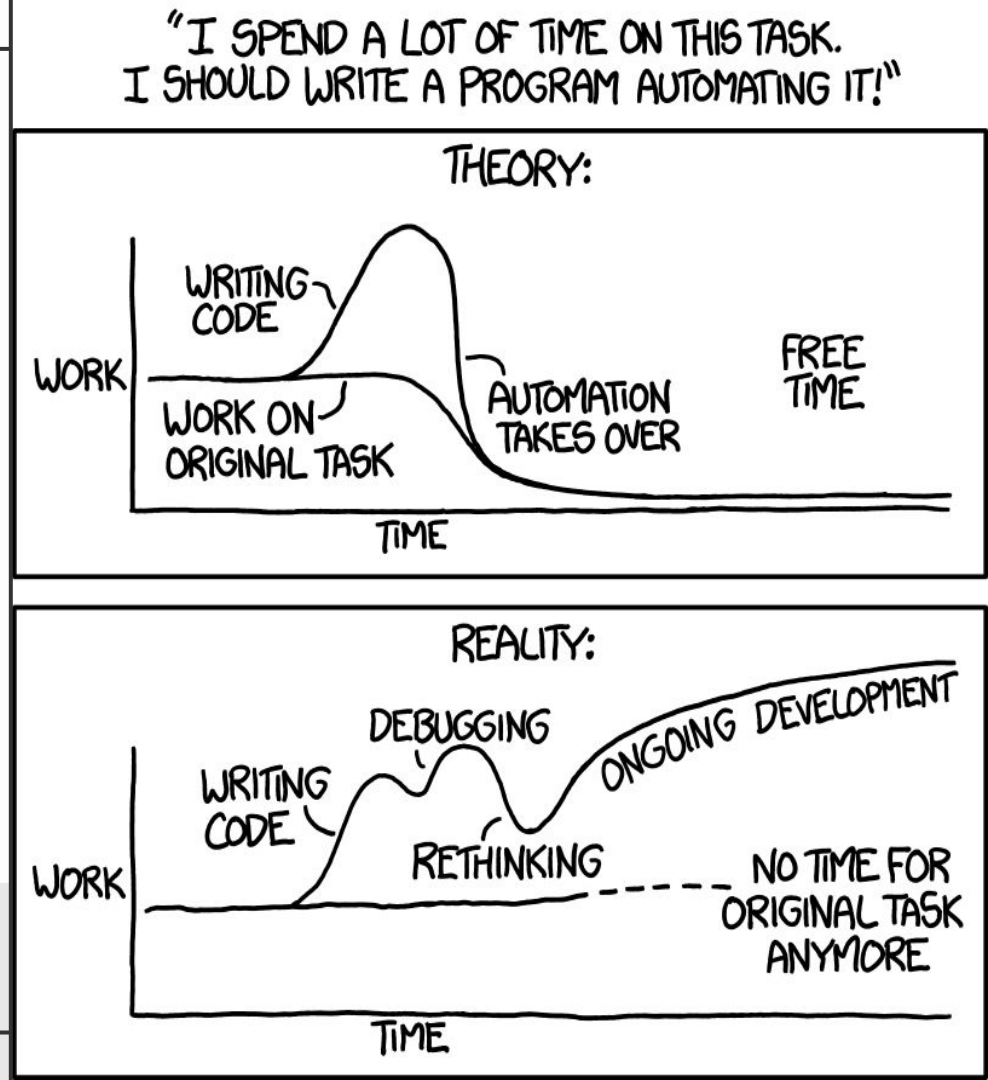
Web Development

Alternatives to the traditional web tech stack include libraries and frameworks that Python can provide.



Automation

The key use of programming is to automate the boring tasks to make it faster and easier.



Specialist Fields

Python is a common entry-point for specialists to build and process their knowledge base.

Python History

Origins

Python was created by Guido van Rossum in 1991 and released in 1994 (version 1.0) when was working the ABC Programming Language Group at the National Research Institute for Math and Computer Science in the Netherlands.

Fun Fact # 1

The name Python was inspired by the BBC's TV Show: Monty Python's Flying Circus

Fun Fact #2

Java's first version was released in 1995 by James Gosling, making Python older.

Python History

Python 1.x

Development started in 1991, but was officially released in January 1994. It was a part of Rossum's Computer Programming for Everybody (CP4E) initiative.

Python 2.x

First instance released in October 2000, under a new license (Python Software Foundation License). This has been deprecated since January 2020.

Python 3.x

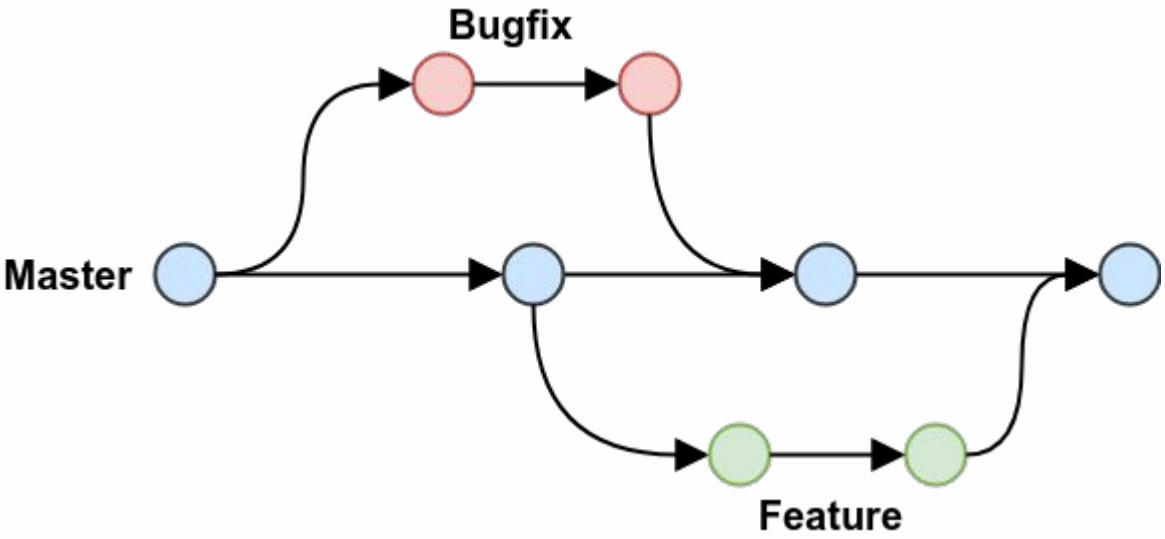
First version released in December 2008, made to upgrade performance, add extra features, and improve clarity without being backwards compatible

Version Control

Taught in the context of git

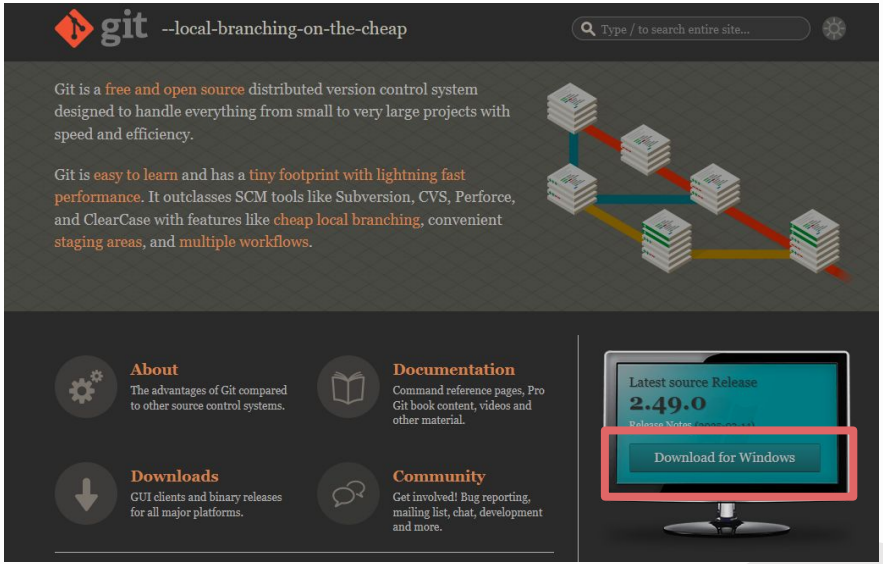
Git

`Git` is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.



Step 1: Download Git

Go to https://git-scm.com/ and select the download option. This is automatically set to your OS.



Step 2: Git Installation Setup

Run the git installer and use the default options ***except for the terminal emulator (for convenience)***

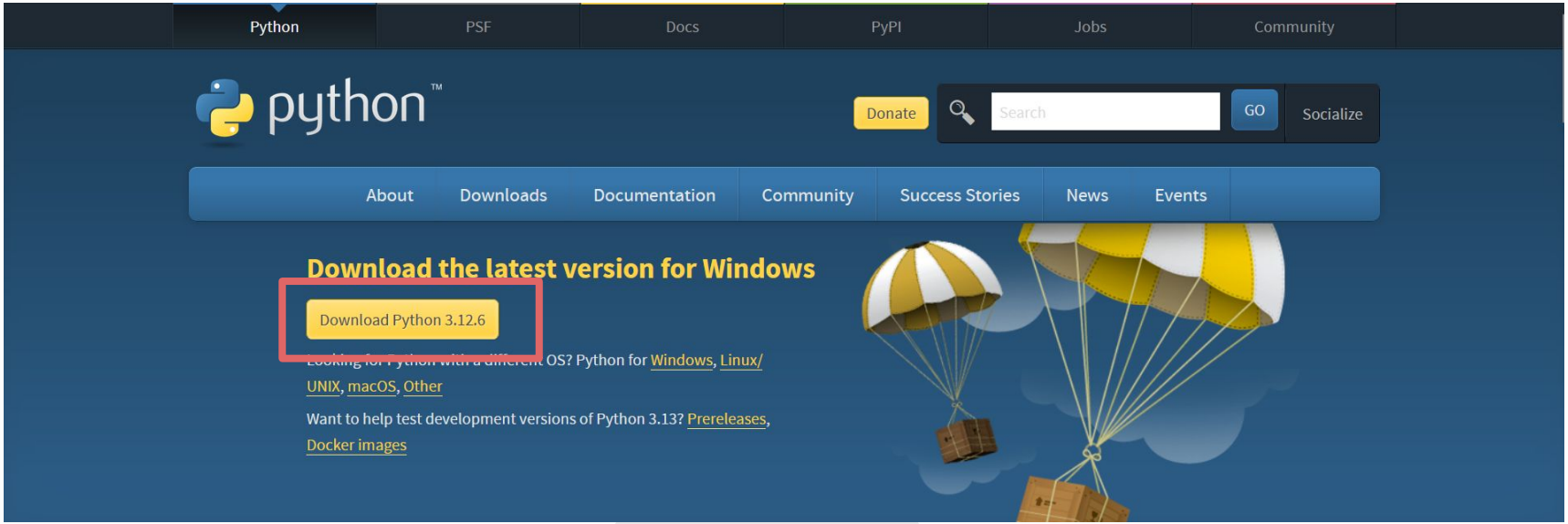
Default Editor	Keep as is
Initial Branch Name	Keep as is
Path Setup	Keep as is
SSH Executable	Keep as is
HTTPS Transport Backend	Keep as is
Checkout	Keep as is
Terminal Emulator	**Use Windows Default Console Window**
Git Pull	Keep as is
Credential Helper	Keep as is
Extra Options	Keep as is

Setup Python

Preparing our Integrated Development Environment (IDE)

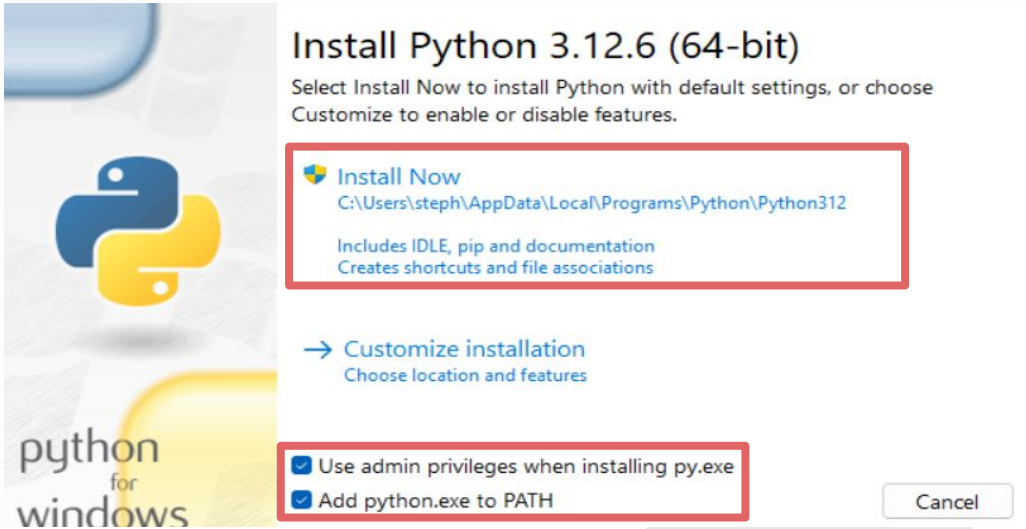
Step 3: Download Python Downloader

Go to **`https://www.python.org/downloads/`** and click the first download button. The version and type is automatically the latest. ***Always opt for the latest version whenever possible.***



Step 4: Run Python Installer

- Run the downloaded installer (preferably with admin privileges)
- Enable all of the checkbox options
- Select Install Now option

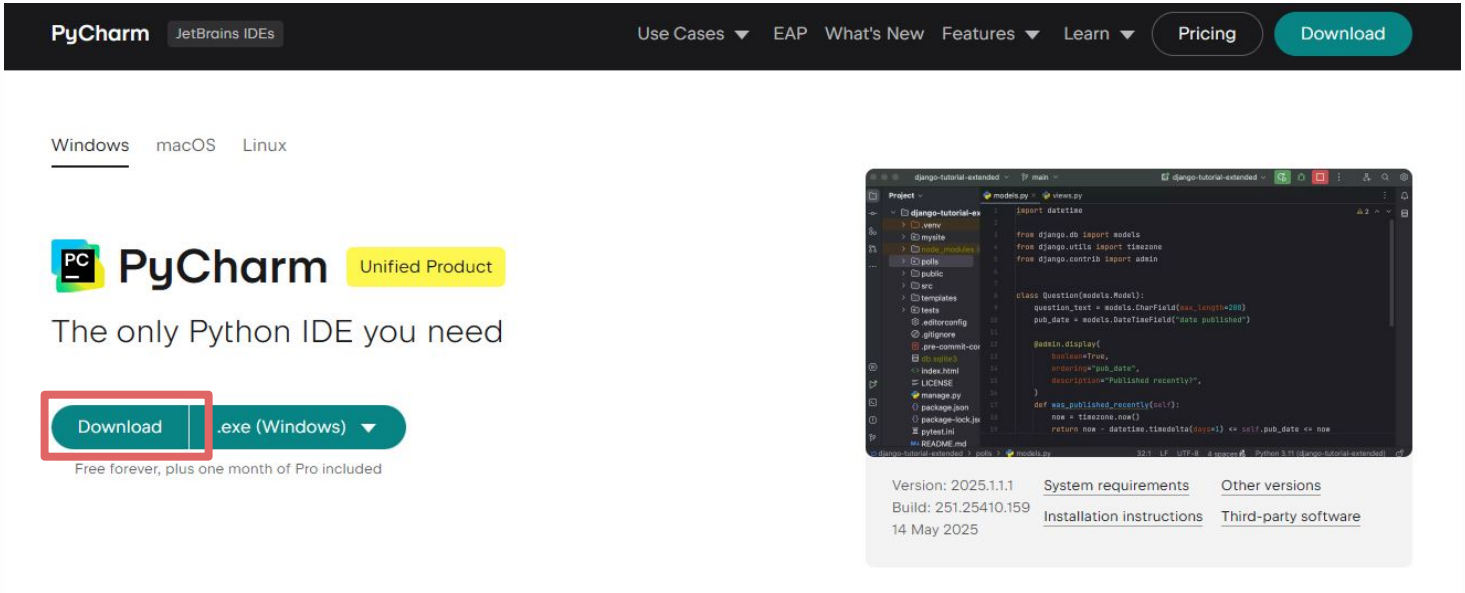


Setup Pycharm

Preparing our Integrated Development Environment (IDE)

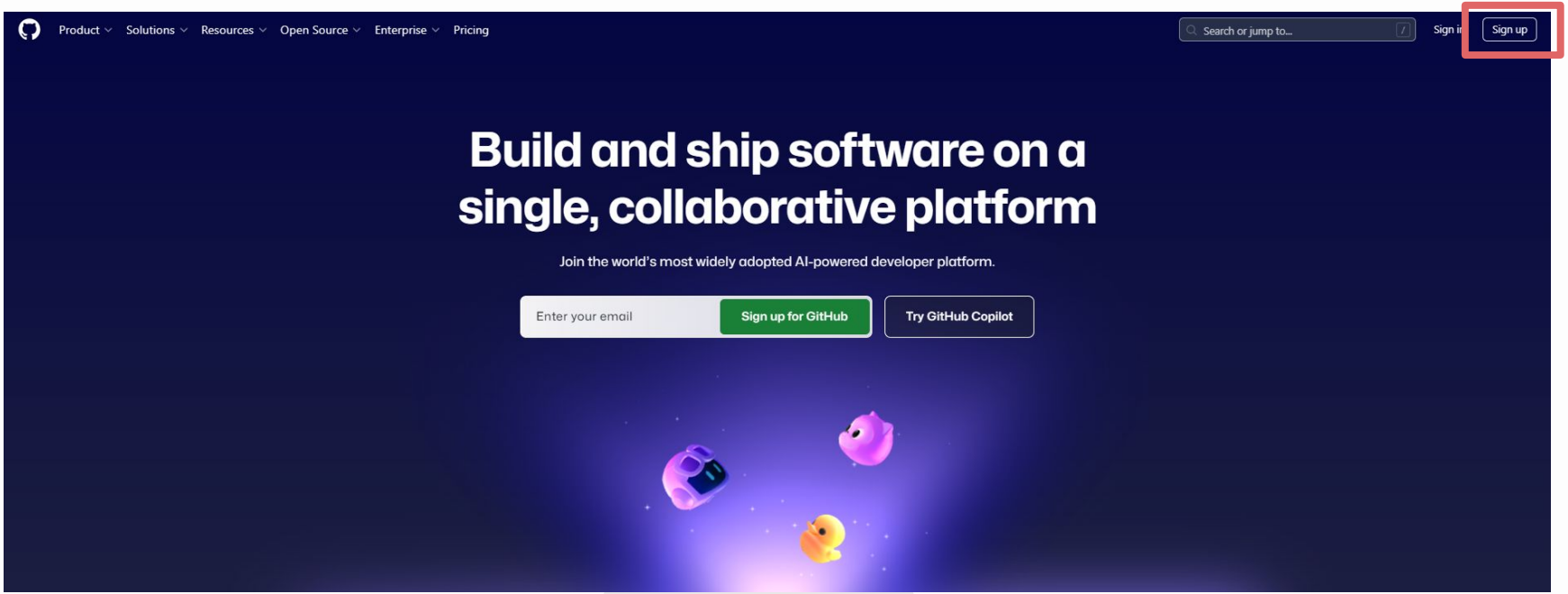
Step 5: Download and Run PyCharm Installer

Go to **`https://www.jetbrains.com/pycharm/download/`** select Download. Afterwards, run the installer and use the default options.



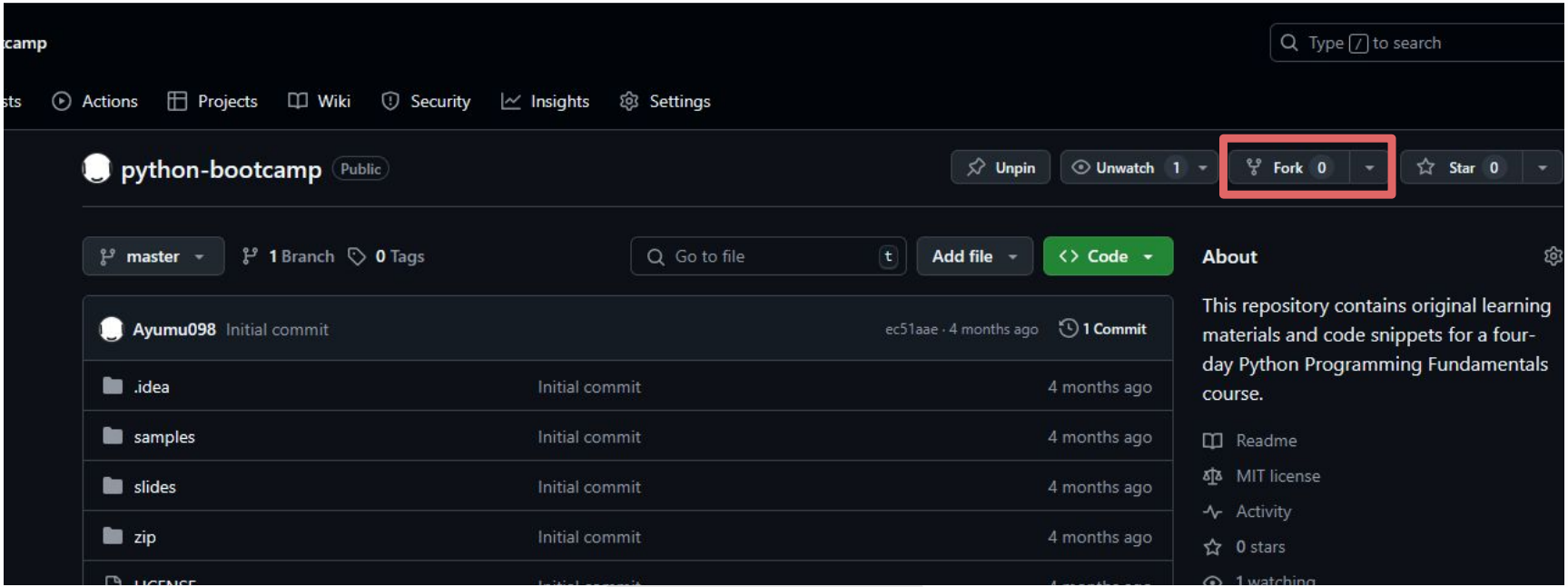
Step 7: Setup Github Account

Go to **`github.com`** and Sign In if you already have an account or sign up for a new account.



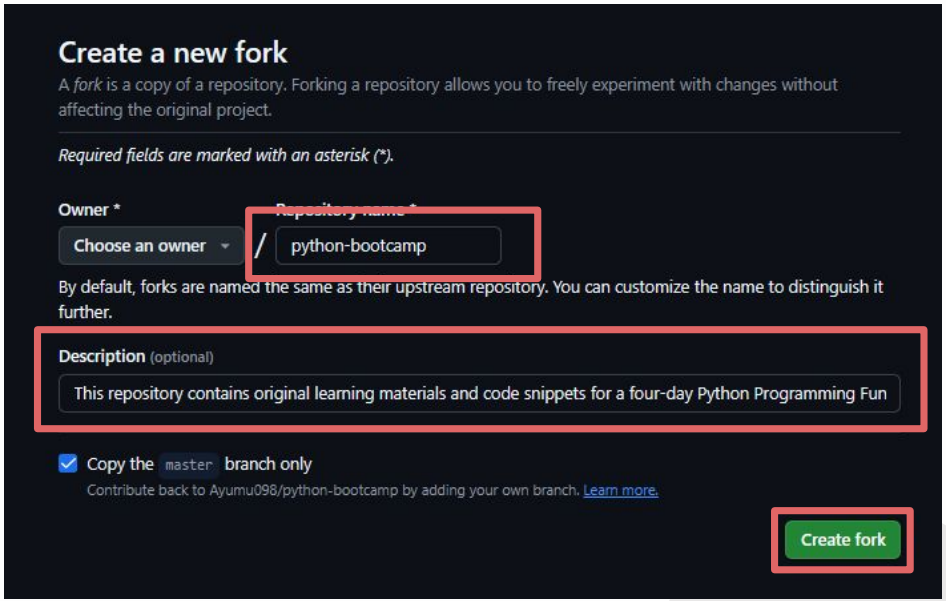
Step 8: Fork python-bootcamp repository

Go to **`github.com/Ayumu098/python-bootcamp`** and select the fork button on the upper right



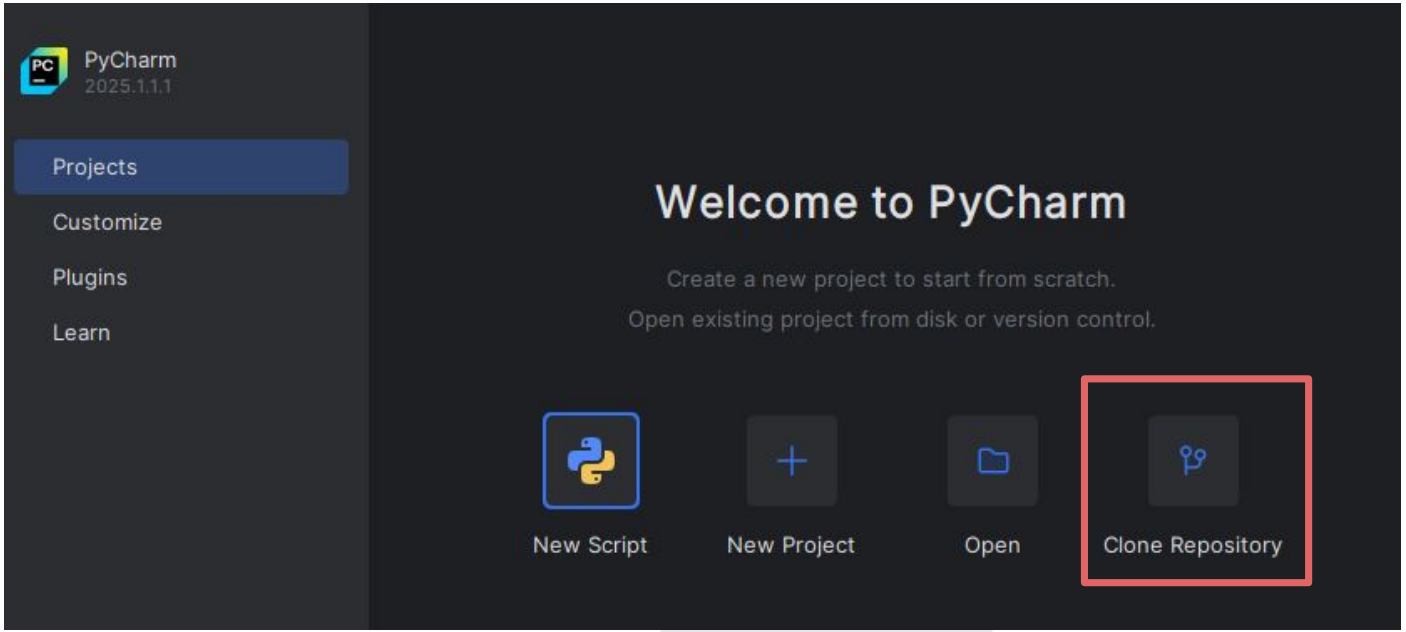
Step 9: Setup Fork Settings

You can change the repository name, owner, and description (optional).
Select your account for owner and then select the **Create Fork** button



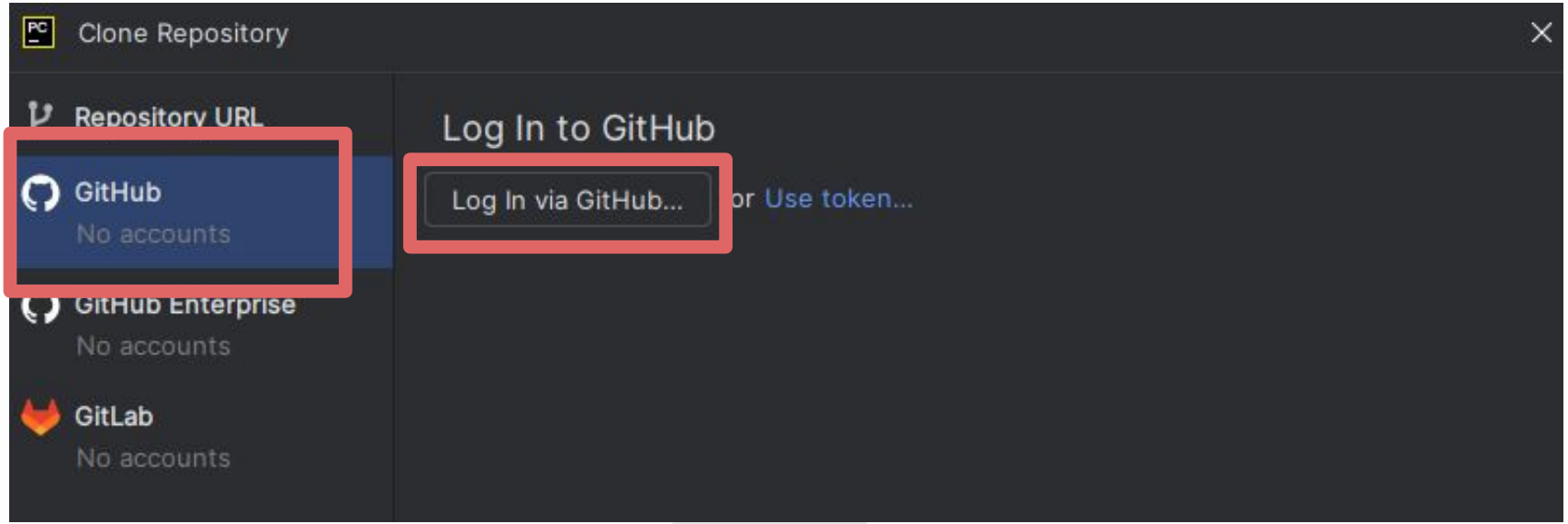
Step 10: Clone Repository

Run PyCharm Community (if not already open) and select the **Clone Repository** option.



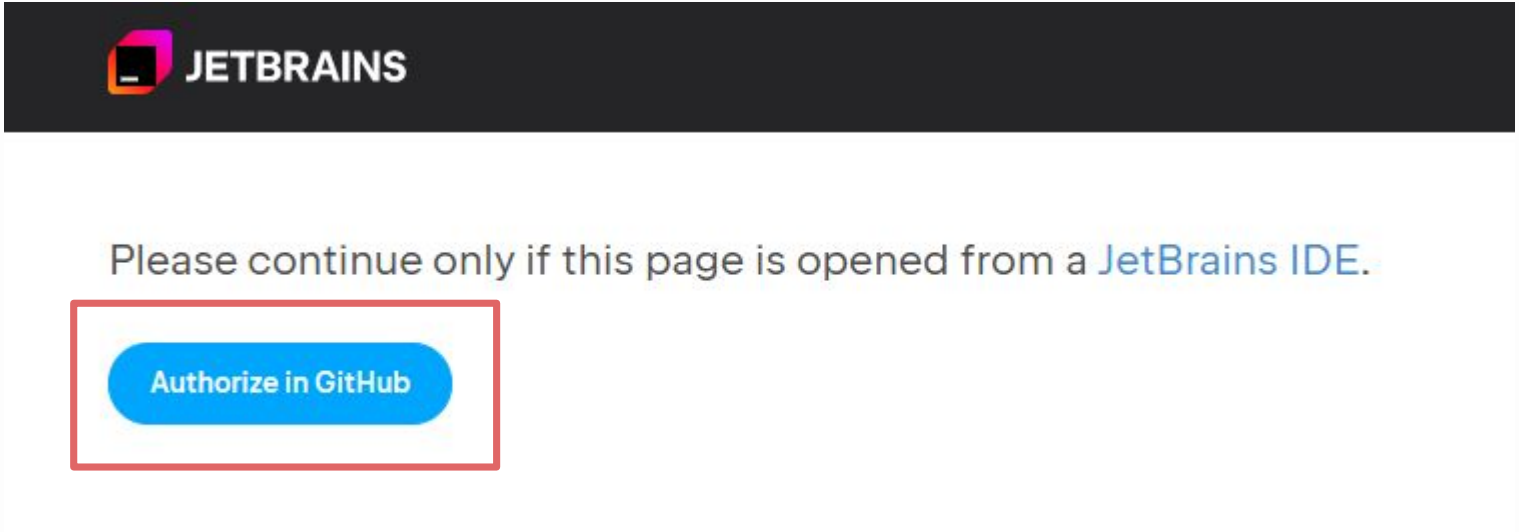
Step 11: Login GitHub for PyCharm

On prompt, select GitHub and log in for credentials



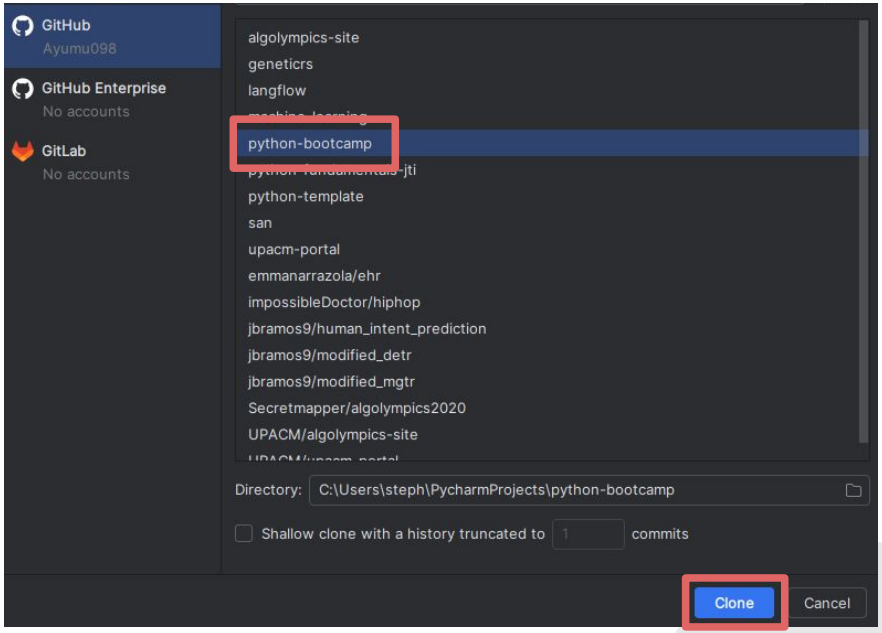
Step 12: Authorize GitHub - PyCharm connection

A new tab will open in the browser. Select **Authorize in Github**



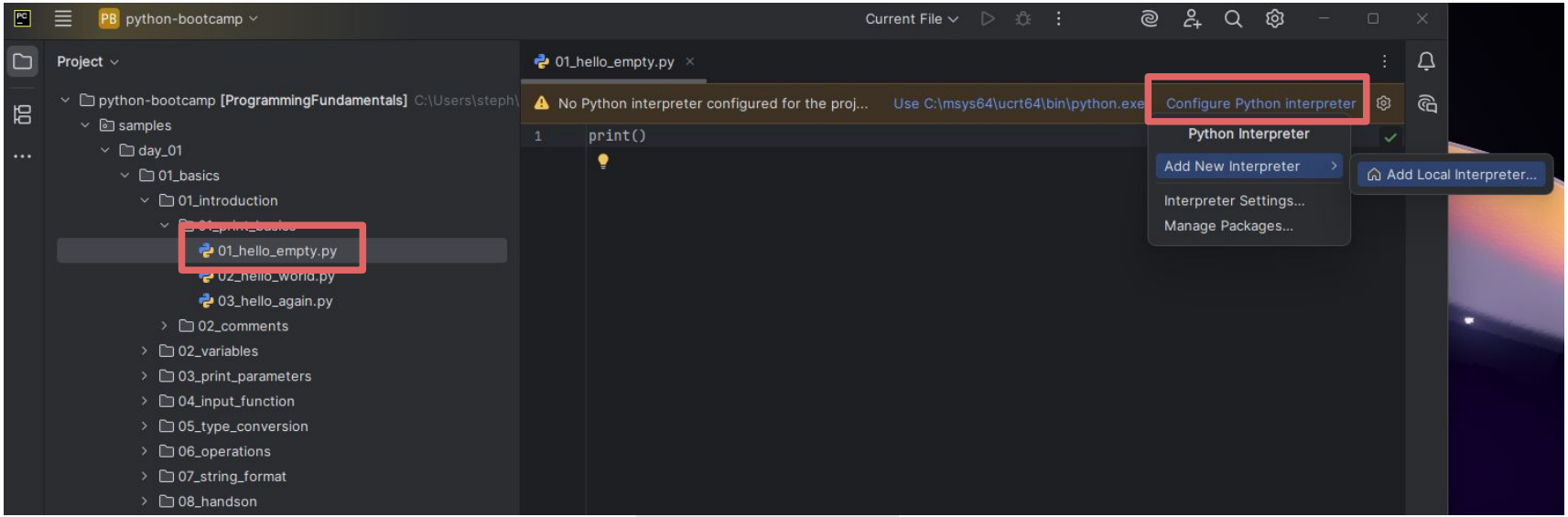
Step 13: Select Repository

The window shows all your repositories. Select the fork and **Clone**.



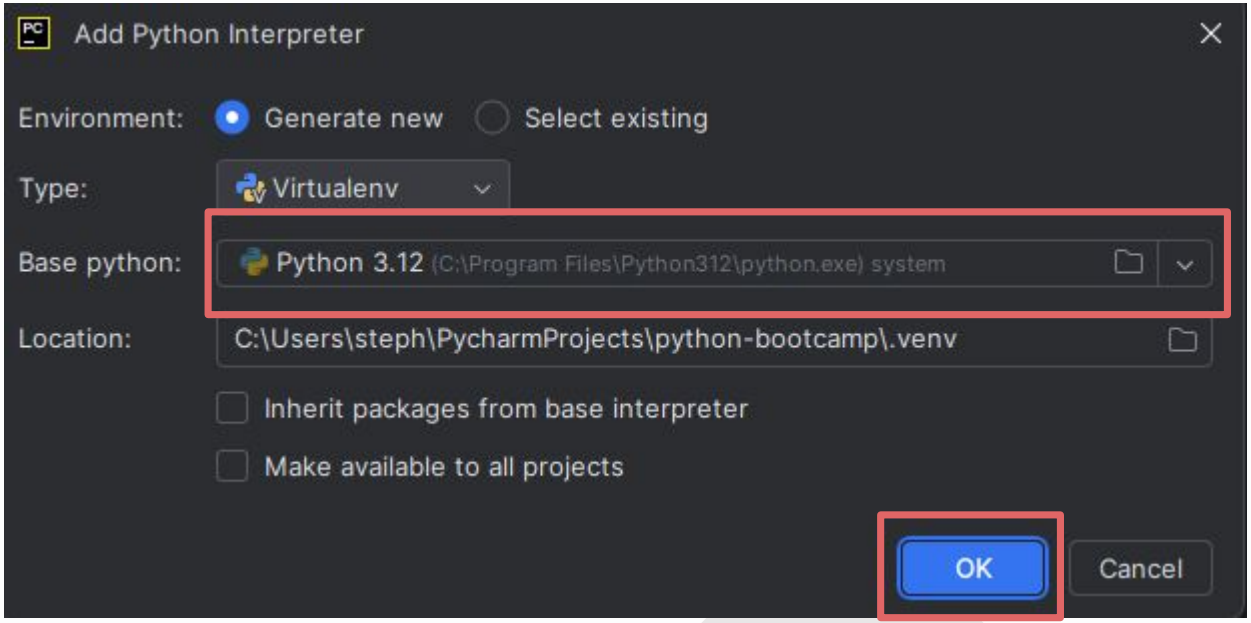
Step 14: Add Python Interpreter

Go to **samples > day_01 > 01_basics > 01_introduction > 01_print_basics > 01_hello_empty.py**.
A yellow warning will appear when the file is opened.



Step 15: Setup Python Interpreter

On prompt, double check if Base Python is already setup and select **OK**.

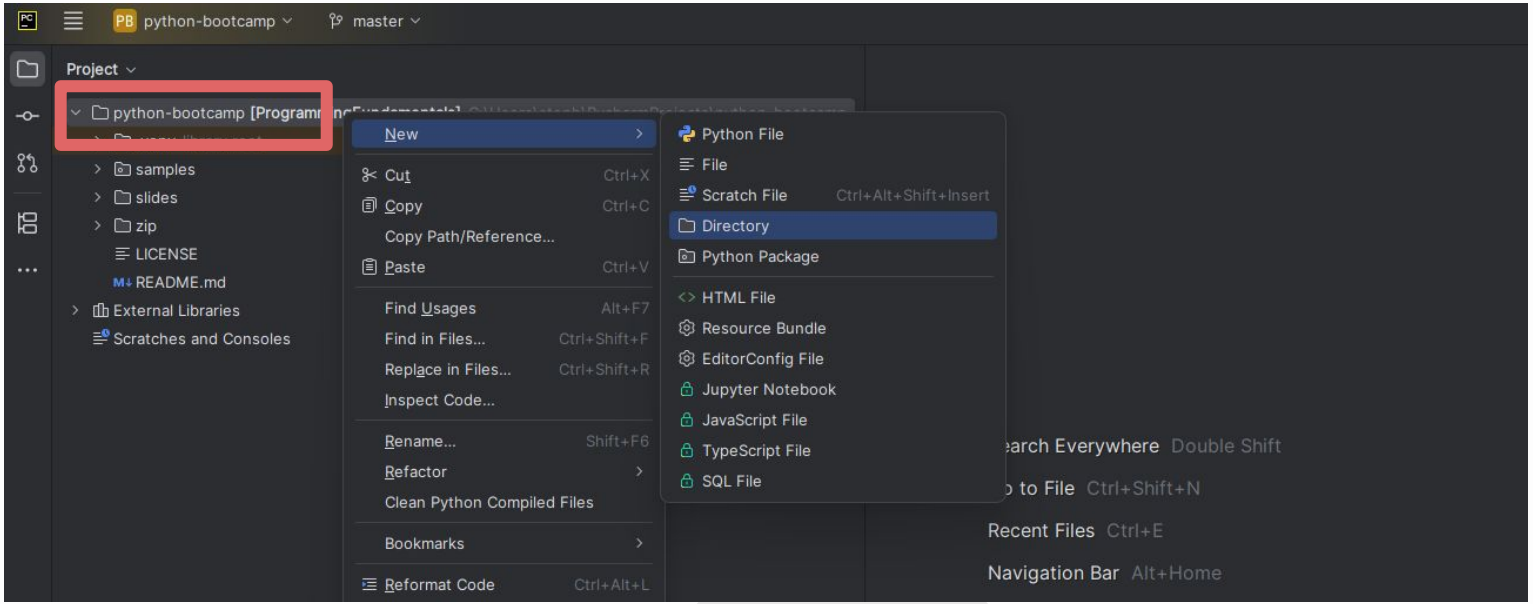


Hello World

A journey of a thousand miles begins with a single step

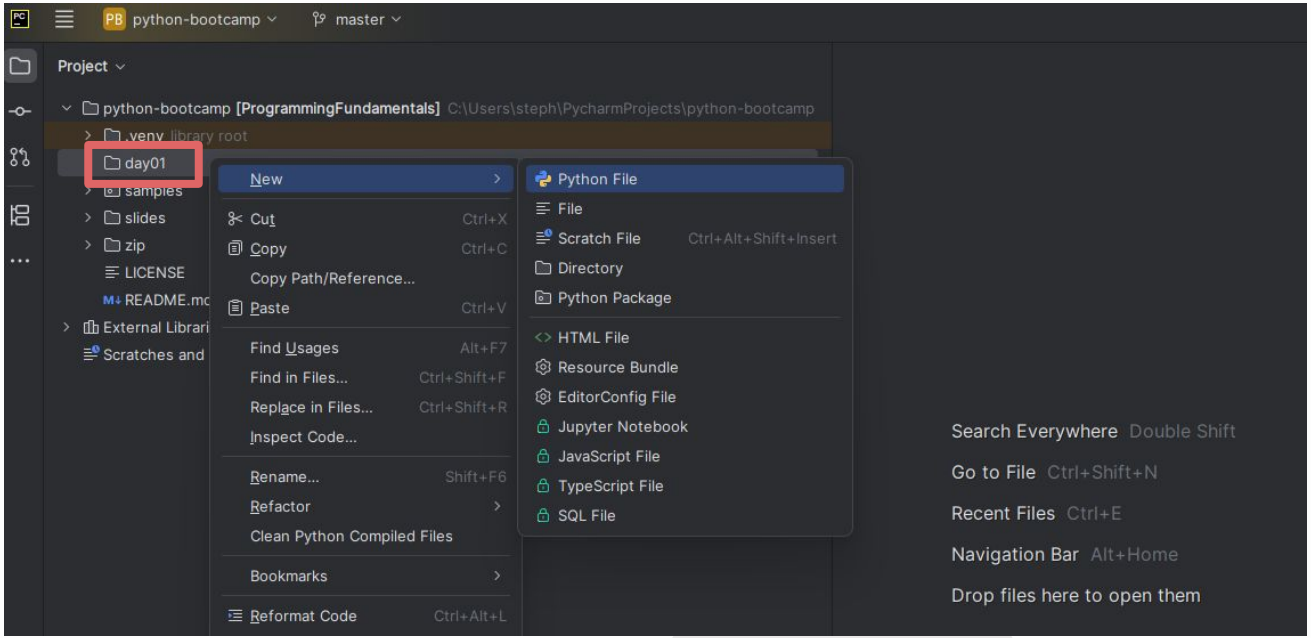
Create a New Folder

Right click the current project folder name, select **`New > Directory`**. Name it **`day01`**



Create New Python File

Right click the new folder name, select **`New > Python File`**. Name it **`01_hello`**



Writing your First Code

print ()

Function	Predefined commands or actions
Parentheses	Marker where `function input` starts and ends

Writing your First Code

```
print (Hello World )
```

Function	Predefined commands or actions
Parentheses	Marker where `function input` starts and ends
Text	

Writing your First Code

```
print ("Hello World" )
```

Function	Predefined commands or actions
Parentheses	Marker where `function input` starts and ends
Text	
Double Quote	Marker where the `text` starts and ends

Multi-line Printing

```
print("Hello World" )
print("Hello Again!" )
```

Single-line Printing

```
print("I", "am", "happy")
```

Quick Exercise: Hello World

Print the following in the console:

01_hello.py

```
Hello! My name is your name
I am learning Python
```

Single-Line Comment

To write a line without being detected as code, use a hashtag on the left side

```
1  print("Hello World")
2  print("Hello Again")
```

```
1  # print("Hello World")
2  print("Hello Again")
```

Multiple-Line Comment

To write multiple lines without being detected as code, use triple quotes at the start and end

```
"""
print("Hello World")
print("Hello Again")
"""
```

Comments for Documentation

Comments are usually used to describe, explain, or justify code

```
1  # Practice for printing in multiple lines
2  print("Hello, I am new to Python")
3  print("Let's learn together!")
```

02

Variables

Representing and storing data in Python

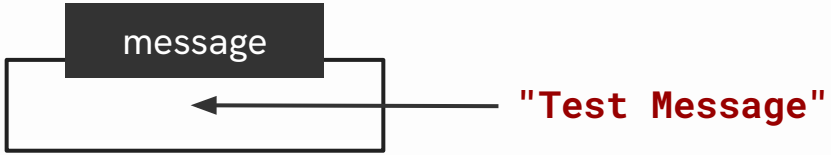
Variable Define

Creating data and storing them

Variable Declaration

A variable can be created by writing its name, the equal sign, and the value

```
message = "Test Message"
```



Variable Naming

Case Sensitive

Variables that differ even by one letter or casing are not the same

No Special Chars

It only supports alphabetical letters or symbols and underscores

Can do Numbers

But it must not be the first part of the variable

Quiz: Is this valid?

```
correct = "True"
```

```
years taken beforehand = 12
```

```
_hidden = "Please keep this a secret"
```

```
$var = 123
```

```
million_dollars = 1000000.00
```

```
何でもない = ""
```

Variable Printing

Variables can also be displayed on the console with the **`print`** function

```
1  message = "Test Message"
```

```
2  print(message)
```

Variable as Nicknames

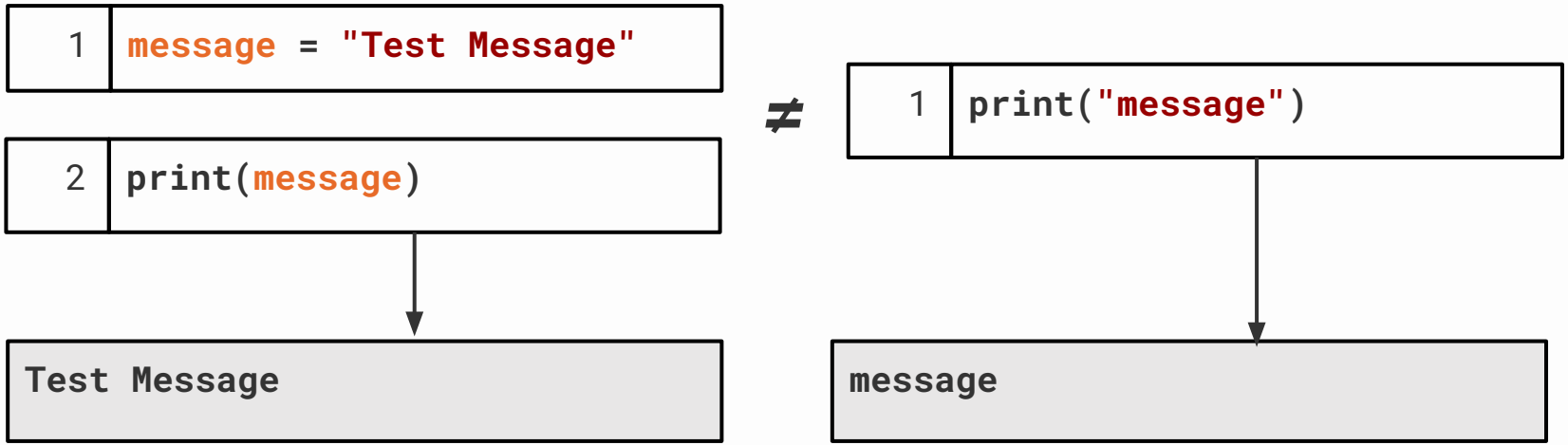
Variables are often used to represent data concisely


```
print(name)
```

```
José Protacio Rizal Mercado y Alonso Realonda
```

Variables and Text

Be careful not to confuse strings and variables (no quotes)



Change earlier code

Old Code:

01_hello.py

```
print("Hello! My name is your name")
print("I am learning Python")
```

New Code:

01_hello.py

```
name = "your name"
language = "Python"

print("Hello! My name is", name)
print("I am learning Python!", language)
```

Variable Change

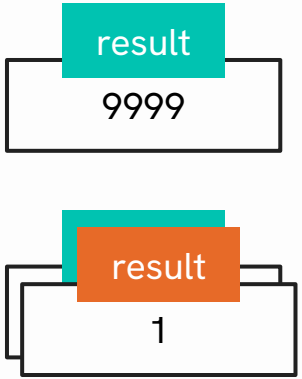
Updating existing variables

Variable Reassignment

Variables can be changed by using the same variable name

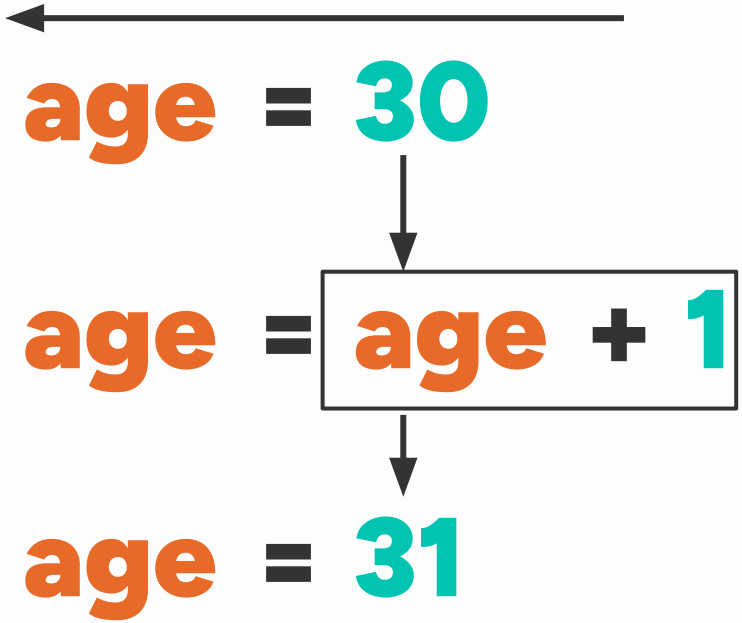
```
result = 9999
print(result)

result = 1
print(result)
```

```
9999
1
```

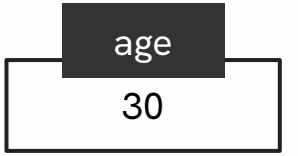


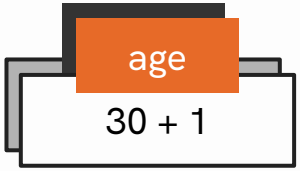
Variable Evaluation

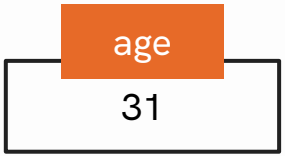
Python evaluates code top to bottom, right to left




```
age = 30
age = age + 1
```







Example 1: Level-up

Using a variable on the right side means using its current value ***with some change***

```
1  level = 1
2  print("Level:", level)
3
4  # Player gains XP
5  level = level + 1
6  print("Leveled up! Level:", level)
```

Example 2: Battery Level

Variables can be reassigned for as long as they are in scope (more on this later)

```
1  battery = 100
2  print("Battery:", battery)
3
4  # Opened Chrome with 10 tabs
5  battery = battery - 40
6  print("After Chrome:", battery)
7
8  # Plugged in charger
9  battery = battery + 20
10 print("Charger inserted:", battery)
```

Quick Exercise: Score System

02_counter.py

```
1  counter = 0
2  print("Counter:", counter)
3
4  # Point up: Add one to the counter
5  # Code here
6  print("Counter:", counter)
7
8  # Bonus: Multiply the score by 10
9  # Code here
10 print("Counter:", counter)
11
12 # Penalty: Decrease the score 4
13 # Code here
14 print("Counter:", counter)
```

Variable Types

The basic forms of data available in Python

Strings (str)

Strings represent text or a series of characters, enclosed in double or single quotes

```
empty_string_a = ''
```

```
empty_string_b = ""
```

```
quote = "I am a little teapot, short and spout."
```

What are other examples of strings?

Integers (int)

Integers represent whole (no decimal), positive, or negative numbers

```
savings = 0
```

```
balance = -100
```

```
high_score = 111_222_333
```

What are other examples of integers

Floating-Point Numbers (float)

Floats represent real positive, or negative numbers with decimal points

```
temperature_celsius = 0.0
```

```
growth_rate = -0.56
```

```
avogadros_number = 6.022e+23
```

What are other examples of floats

Boolean (bool)

Booleans represent True or False

```
is_raining = True
```

```
exit_program = False
```

What are other examples of booleans?

None (None)

None represent null or empty values

```
response = None
```

What are other examples of None?

Quick Exercise: Favorite Sample

02_favorite_sample.py

```
1  # Create the following variables and fill in the values
2  # Make sure to use the correct data type!
3  favorite_food_name = Your favorite food's name
4  favorite_food_price = How much does it usually cost to make it?
5  is_homemade =  Is it a type of food that's homemade?
6
7  # Then, print each information one line at a time
8  print(favorite_food_name)
9  print(favorite_food_price)
10 print(is_homemade)
```

Print Parameters

Extra features for printing data in the console

Print Separator Parameter

The `print()` function also accepts special inputs. One of these is `sep` that defines the separator between multiple inputs.

```
1 print(1, 2, 3, 4, sep=" | ")
```

```
1 | 2 | 3 | 4
```

```
2 print(1, 2, 3, 4, sep="")
```

```
1234
```

Quick Exercise: Succession

03_succession.py

```
1  # From: 1 2 3 4 5 6
2  # To:   1 -> 2 -> 3 -> 5 -> 6
3  print(1, 2, 3, 4, 5, 6)
```

Print End Parameter

The `end` parameter defines the final string added (the default is newline `"\n"`)

```
1  print("Items", end=": ")
2  print("Chocolate", end=", ")
3  print("Strawberry", end="!")
```

```
Items: Chocolate, Strawberry
```

Escape Characters

To add special characters in Python strings, use an escape key `\`

Character	Symbol
Single Quote	`\"`
Double Quote	`\'`
Backslash	`\\`
Tab	`\t`
Newline	`\n`

Print End Parameter

The `end` parameter defines the final string added (the default is newline `"\n"`)

```
1  print("Items:", end="\n\t")
2  print("Chocolate", end="\n\t")
3  print("Strawberry")
```

```
Items:
    Chocolate
    Strawberry
```

Quick Exercise: Loading

04_unending.py

```
1  # Change the code to print each part in a single line
2  print("Loading")
3  print(".")
4  print(".")
5  print(".")
```

Input Function

Using data given by the user of the code

Input Function

The **input** function gets data given in the console. The given data can then be stored in a variable. Note: The input will always return a str.

```
1  user_input = input()
2  print(user_input)
```

Input Function (With Instructions)

The input function accepts a single parameter. This is used to print a message before getting user input (for additional context).

```
1 user_input = input("Enter input: ")
2 print(user_input)
```

Return Values

The input function can still work if the user data is not stored in a variable. However, the data will not be accessible for later use.

```
1  input("Enter input: ")
2  print()
```

Quick Exercise: Favorite Sample (version 2)

02_favorite_sample.py

```
1  # Create the following variables and fill in the values
2  # Make sure to use the correct data type!
3  favorite_food_name = Your favorite food
4  favorite_food_price = Cost of your favorite food
5  is_homemade =  Is the food homemade?
6
7  # Then, print each information one line at a time
8  print(favorite_food_name)
9  print(favorite_food_price)
10 print(is_homemade)
```

Type Conversion

Making operations and processes compatible with Python

User Input Issue

What is the problem with this code?

```
number_input = input("Enter number: ")
print(number_input + 1)
```

Strings and Numbers

Be careful not to confuse integers and strings that look like integers

```
1  number = "123"
2  number = number + 1
```

≠

```
1  number = 123
2  number = number + 1
```

Integer Type Conversion

You can convert most basic data types to int with the `int()` function. Try the following:

```
number_input = input("Enter number: ")
print(number_input + 1)
```

```
number_input = int(input("Enter number: "))
print(number_input + 1)
```

Funny Issue: Number Adder

What is the problem with this code?

```
# Ask the user for two numbers
first_number = First Number
second_number = Second Number

# Then, print the sum
sum = first_number + second_number
print(sum)
```

Quick Exercise: Number Adder

05_number_adder.py

```
1  # Ask the user for two numbers
2  first_number = First Number
3  second_number = Second Number
4
5  # Then, print the sum
6  sum = first_number + second_number
7  print(sum)
```

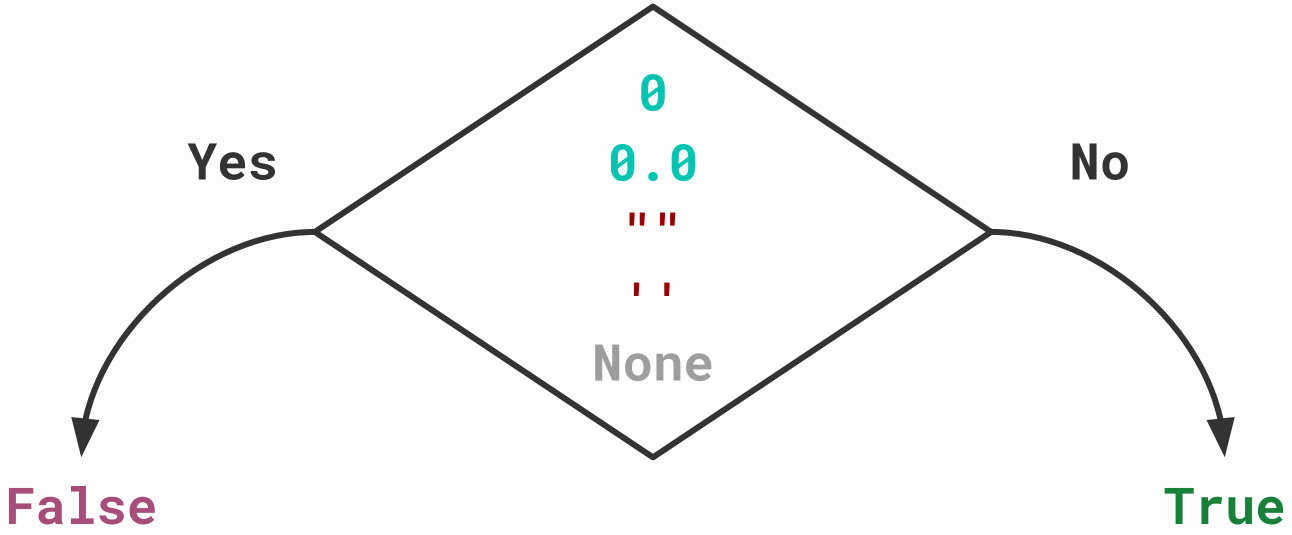
Integer Type Conversion

Original Data Type	Result
Float	Drops all decimal places
Boolean	True →1, False → 0
String	Converts to integer. If invalid, **raises an error**
None	**Raises an error**

Float Type Conversion

Original Data Type	Result
Integer	Adds .0 decimal place
Boolean	True → 1.0, False → 0.0
String	Converts to float. If invalid, **raises an error**
None	**Raises an error**

Boolean Type Conversion



Boolean Type - Common Pitfalls

Value	Notes
`" "`	Spaces are not completely empty → **`True`**
`' '`	Spaces are not completely empty → **`True`**
`"False"`	Non-empty string → **`True`**
`"None"`	Non-empty string → **`True`**


Typing

What is the final data type and value?

```
name = "Jeff"
print( str(name ))
```

What is the final data type and value?

```
pi = 3.14
print( int(pi))
```

What is the final data type and value?

```
coffee_cups = "2"
print( int(coffee_cups ) + 1)
```

What is the final data type and value?

```
taho_price  = 15
print( str(taho_price ) + " pesos" )
```

What is the final data type and value?

```
wifi_speed  = 4.9
print( int (wifi_speed ))
```

What is the final data type and value?

```
world = "hello"
print( int(world ))
```

What is the final data type and value?

```
crush_replied  = "False"
print( bool (crush_replied ))
```

What is the final data type and value?

```
gcash_balance  = "0"
print( bool (int (gcash_balance  )))
```

What is the final data type and value?

```
x = False
print( str(int(x)))
```

Operations

Applying transformations to data

Number Operations

Symbol	Operation	Example	Result
+	Addition	`result = 11 + 2`	13
-	Subtraction	`result = 11 - 2`	9
*	Multiplication	`result = 11 * 2`	22
/	Division	`result = 11 / 2`	5.5

Floor Division

The floor division operator divides the number on the left by the right and rounds down

```
1  division = 11 / 2
2  print(division)
3
4  floor_division = 11 // 2
5  print(floor_division)
```

```
5.5
5
```

Exponent/Power Operator

The exponent operator multiplies the number on the left by itself multiple times

```
1  result = 2 ** 4
2  print(result)
3
4  manual_result = 2 * 2 * 2 * 2
5  print(manual_result)
```

```
16
16
```

Modulo/Remainder Operator

The modulo operator returns the remainder if the left side was divided by the right side

```
1  result = 11 % 3
2  print(result)
```

```
2
```

$$11 \div 3 \quad = \quad 3 \text{ remainder } 2$$

$$\begin{aligned} &\text{Step 1:} \quad 3 \times 3 = 9 \quad && (3 \text{ fits into } 11 \text{ three times}) \\ &\text{Step 2:} \quad 11 - 9 = 2 \quad && (\text{remainder is } 2) \end{aligned}$$

Order of Operations

Given the following operation:

$$3 + 5 \times 2 - \frac{8}{4}$$

This can be translated in Python with the following expression:

```
result = 3 + 5 * 2 - 8 / 4
print(result)
```

```
result = 3 + (5 * 2) - (8 / 4)
print(result)
```

Quick Exercise: Download Time

06_download_time.py

```
speed = float(input("Download Speed (Mbps): "))
file_size = float(input("File size (MB): "))

# MegaBytes per second (MBps) = 8 * Megabits per second (Mbps)
download_speed_MBps = None

# Download Time (Seconds) = Size (MB) / MegaBytes per second (MBps)
download_time_seconds = None

# Download Time (Minutes) = Download Time (Seconds) / 60
download_time_minutes = None

print(download_time_minutes)
```

String Concatenation (Addition)

Multiple strings can be combined using the addition operator

```
1 print("Hello" + " " + "Hello")
```

```
Hello World
```

String Repetition (Multiplication)

A string can be repeated multiple times using the multiplication operator.

```
1 print("ice " * 3)
```

```
ice ice ice
```

Challenge: Ice Ice Ice Baby

07_ice_ice_ice_baby.py

```
ice = "Ice"
baby = "Baby"

# Print "Ice Ice Ice Baby" using + and *
print()
```

Update Shortcut

All operations where the variable is changed by a copy of it can be simplified

Original Statement	Shortcut
`result = result + 5`	`result += 5`
`result = result * 10`	`result *= 10`
`message = message + "World"`	`message += "World"`

Example 1: Level-up (Updated)

Using a variable on the right side means using its current value ***with some change***

```
1  level = 1
2  print("Level:", level)
3
4  # Player gains XP
5  # Previously: level = level + 1
6  level += 1
7  print("Leveled up! Level:", level)
```

Example 2: Battery Level (Updated)

Variables can be reassigned for as long as they can be accessed

```
1  battery = 100
2  print("Battery:", battery)
3
4  # Opened Chrome with 10 tabs
5  # Previously: battery = battery - 40
6  battery -= 40
7  print("After Chrome:", battery)
8
9  # Plugged in charger
10 # Previously: battery = battery + 20
11 battery += 20
12 print("Charger inserted:", battery)
```

Quick Exercise: Score System (version 2)

02_counter.py

```
1  counter = 0
2  print("Counter:", counter)
3
4  # Point up: Add one to the counter
5  # Change your code in this line
6  print("Counter:", counter)
7
8  # Bonus: Multiply the score by 10
9  # Change your code in this line
10 print("Counter:", counter)
11
12 # Penalty: Decrease the score 4
13 # Change your code in this line
14 print("Counter:", counter)
```

String Formats

Combine strings and variables conveniently

String Placeholder

Placeholders are represented using curly brackets. They can then be replaced using the `.format()` method or function

```
1  # Message Template
2  message = "Hello {}! Nice to meet you!"
3  print(message)
4
5  # Use Template
6  formatted_message = message.format("Juan")
7  print(formatted_message)
```

```
Hello {}! Nice to meet you!
Hello Juan! Nice to meet you!
```

String Placeholder (Repeated Use)

```
1  # Message Template
2  message = "Hello {}! Nice to meet you!"
3  print(message)
4
5  # Use Template
6  formatted_message = message.format("Juan")
7  print(formatted_message)
8
9  # Use Template (again)
10 new_message = message.format("Jesse")
   print(new_message)
```

```
Hello {}! Nice to meet you!
Hello Juan! Nice to meet you!
Hello Jesse! Nice to meet you!
```

Multiple String Formatting

The format method supports multiple inputs as well as needed.

```
1  message = "Hello {}. Your nickname is {}"
2
3  name = input("Enter name: ")
4  nickname = input("Enter nickname: ")
5
6  formatted_message = message.format(name, nickname)
7  print(formatted_message)
```

Multiple String Formatting (Named)

Placeholders can be given a variable name to make assignment easier

```
1  message = "Hello {first}. Your nickname is {second}"
2
3  name = input("Enter name: ")
4  nickname = input("Enter nickname: ")
5
6  formatted_message = message.format(first=name, second=nickname)
7  print(formatted_message)
```

Quick Exercise: Price Post

08_price_post.py

```
1  # Price notification template
2  price_notification = "The price of {} is ${}."
3
4  # Post: Latte ($3.50)
5  print(price_notification)
6
7  # Post: Espresso ($2.75)
8  print(price_notification)
9
10 # Post: Cappuccino ($4.00)
11 print(price_notification)
```

String Formatting (Modern)

This is the old format that is still used to this day

```
1 name = input("Enter your name: ")
2 print("Hello {} Nice to meet you!".format(name))
```

For short strings, the modern f-string format is used

```
1 name = input("Enter your name: ")
2 print(f"Hello {name} Nice to meet you!")
```

Quick Exercise: Scrapbook Sample

09_scrapbook_sample.py

```
1  # Ask the user for the following inputs
2  your_name = Your name
3  favorite_number = Your favorite number
4  favorite_color = Your favorite color
5
6  # Print: "Your name is your name"
7  print()
8
9  # Print: "Your favorite number is your favorite number"
10 print()
11
12 # Print: "Your favorite color is your favorite color"
13 print()
```

H1

Cost Calculator

Quick practice of all the concepts discussed so far

Cost Calculator

10_cost_calculator.py

```
# Ask the user for the following inputs
item_1_price = Input your item price 1
item_2_price = Input your item price 2
item_3_price = Input your item price 3

# Print: Total Cost
total = None
print(total)
```

Bonus Challenge:

Item Quantity?

03

Control Flow

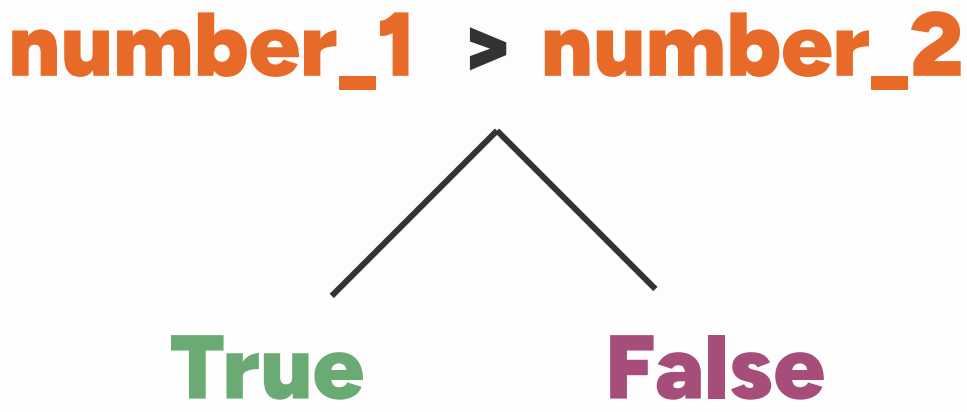
Providing logic to data processing


PASS
FAIL

Relations

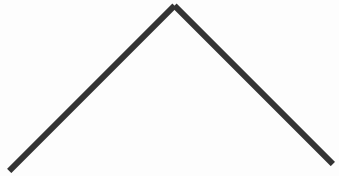
Checking if two values are related to each other

What are the possible results?



The result can be stored

result = number_1 > number_2



True

False

Relational Operator

All of the basic data types (except None) support relational operator (returns a bool)

Symbol	Operation	Example	Value
<	Less Than	11 < 2	False
<=	Less than or Equal	11 <= 2	False
>	Greater Than	11 > 2	True
>=	Greater Than or Equal	11 >= 2	True

Remember: PACMAN First

Quick Exercise: Height Requirement

11_height_requirement.py

```
1  minimum_height = 138
2
3  # Ask the user for the following inputs
4  user_height = User height (in cm)
5
6  # Notify user if they can enter the ride
7  # Change the variable value here
8  can_enter_ride = None
9  print("Can enter the ride:", can_enter_ride)
```

Chained Relational Operator

Similar to the mathematical notation, relational operators can be chained to ask for ranges

```
1  x = int(input("Enter number: "))
2
3  print("Exclusive Range")
4  print(3 < x < 20)
5
6  print("Equal or Greater than 3 and Less than 20")
7  print(3 <= x < 20)
8
9  print("Greater than 3 and Less than or Equal to 20")
10 print(3 < x <= 20)
11
12 print("Inclusive Range")
13 print(3 <= x <= 20)
```

Quick Exercise: Valid Score

12_valid_score.py

```
1  # Range minimum and maximum bounds
2  min_number = 0
3  max_number = 100
4
5  # Enter user input
6  number = Enter number
7
8  # Notify user if the number is a valid score
9  # Change the variable value here
10 valid_score = None
11 print("Valid score:", valid_score)
```

Value (In)equality

The most common relation operator is the equal and not equal operators

Symbol	Operation	Integer Example	String Example
`==`	Equal	`11 == 2`	`"Hello" == "World"`
`!=`	Not Equal	`11 != 2`	`"Hello" != "World"`

Example: Perfect Score Check

Similar to the mathematical notation, relational operators can be chained to ask for ranges

```
1  score = int(input("Enter score: "))
2  perfect_score = 100
3
4  has_perfect_score = score == perfect_score
5
6  print("You got a perfect score:", has_perfect_score)
```

Quick Exercise: Password Check

13_password_check.py

```
1  # Expected password (you can change the value)
2  correct_password = "pass"
3
4  # Enter user password
5  password_input = input("Please provide password: ")
6
7  # Notify user if password is valid
8  # Change the variable value here
9  correct_password_given = None
10
11 print("Access:", correct_password_given )
```

Conditionals

Control when code executes

```
if value1 ? value1 :
    """code"""
```

```
condition = value1 ? value1
if condition :
    """code"""
```

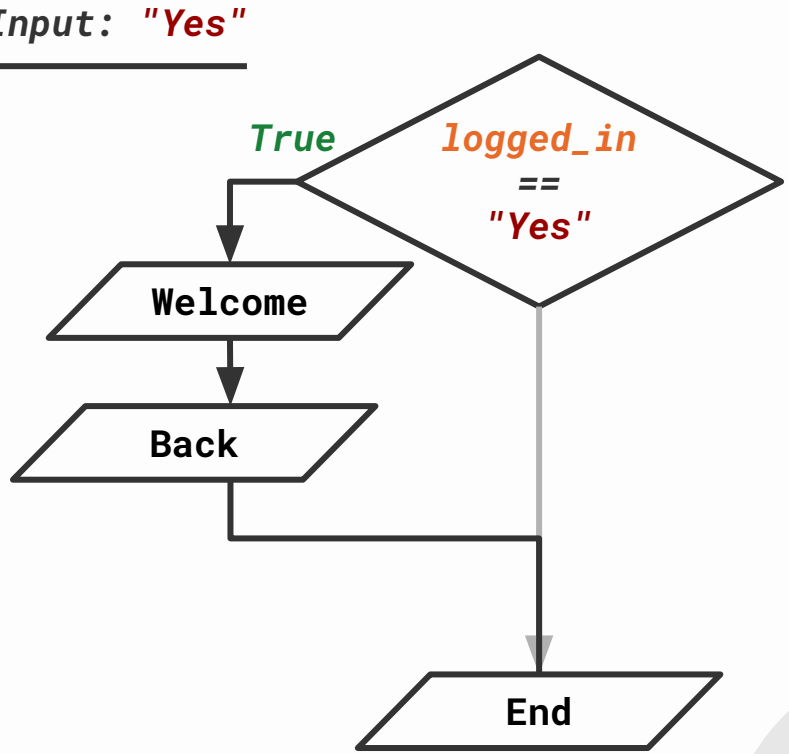
If Statement

```
1  login_input = input("Login: ")
2
3  if login_input == "Yes":
4      print("Welcome")
5      print("Back")
6  print("End")
```

If Statement - True

```
1  login_input = input("Login: ")
2
3  if login_input == "Yes":
4      print("Welcome")
5      print("Back")
6  print("End")
```

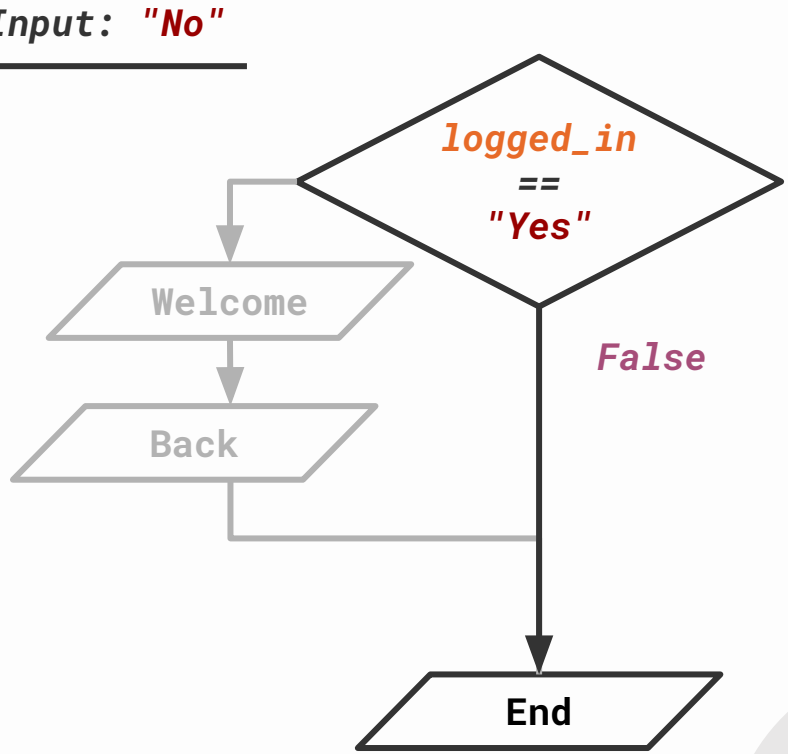
```
Welcome
Back
End
```



If Statement - False

```
1  login_input = input("Login: ")
2
3  if login_input == "Yes":
4      print("Welcome")
5      print("Back")
6  print("End")
```

```
Welcome
Back
End
```



If Statement Example 02

```
1  number = int(input("Enter number: "))
2
3  if number >= 0:
4      print("Number is positive")
```

User enters *"10"*

True

```
Number greater than zero!
```

```
1  number = int(input("Enter number: "))
2
3  if number >= 0:
4      print("Number is positive")
```

User enters *"-1"*

False

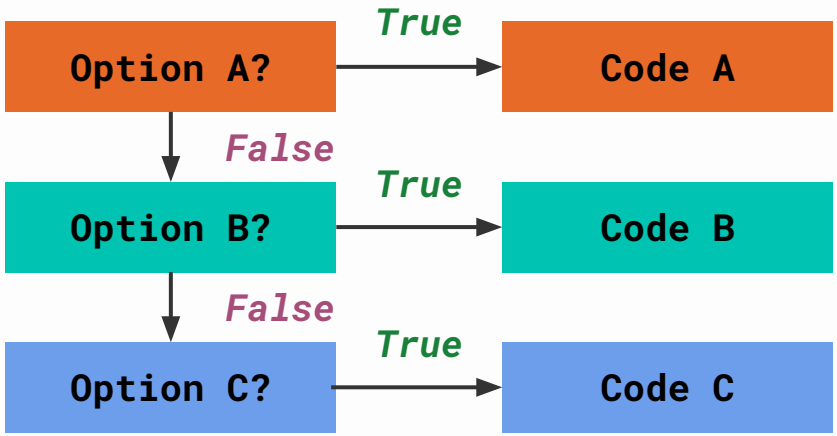
Quick Exercise: Password Check (v2)

13_password_check.py

```
1  # Expected password (you can change the value)
2  correct_password = "pass"
3
4  # Enter user password
5  password_input = input("Please provide password: ")
6
7  # If correct_password_give, print: "Access Granted"
8  # If not correct_password, don't do anything
9  correct_password_given = None
10
11 print("Access Granted")
```

Elif Statement

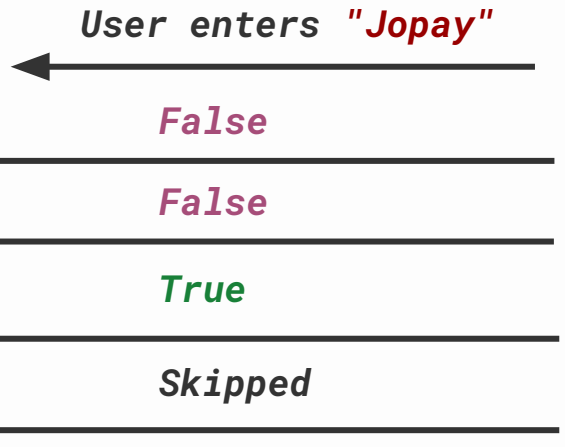
The else-if or elif statements allow you to run parts of the code when the first condition is `False` but there are other possible options




```
if condition_1:
    """Code A"""

elif condition_2:
    """Code B"""

elif condition_3:
    """Code C"""
```

Elif Statement Example 01

```
1  you_said = input("You said: ")
2
3  if you_said == "Wish":
4      print("107.5")
5  elif you_said == "Hello":
6      print("...it's me")
7  elif you_said == "Jopay":
8      print("...kamusta ka na")
9  elif you_said == "Black Pink":
10     print("...in your area")
```




```
...kamusta ka na
```

Elif Statement Example 02

```
1  battery = int(input("Battery percentage: "))
2
3  if battery >= 80:
4      print("Full Battery")
5  elif battery >= 40:
6      print("Good Battery")
7  elif battery >= 15:
8      print("Low Battery")
9  elif battery > 0:
10     print("Critically Low Battery")
11 else:
12     print("No Battery")
```

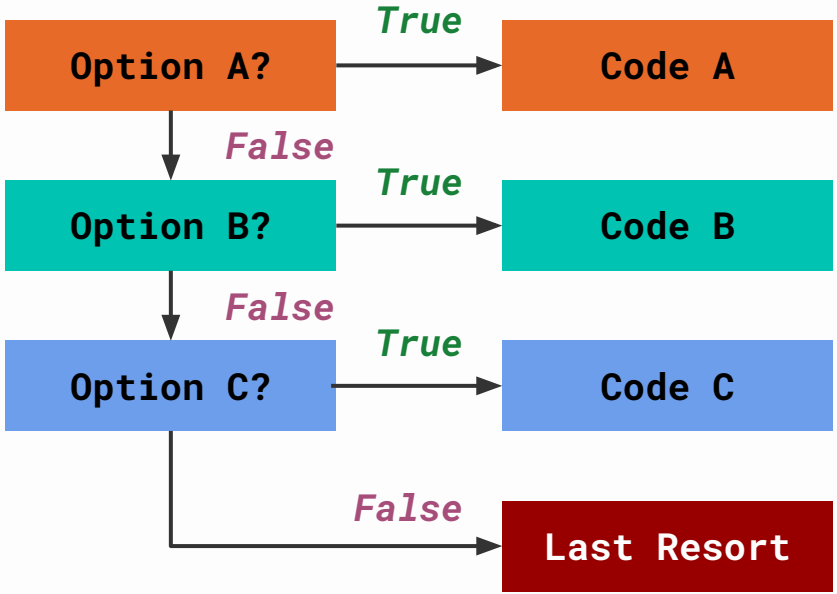
Quick Exercise: Traffic Lights

14_traffic_lights.py

```
1  # Ask the user input for a color
2  color_input = input("Please enter a color: ")
3
4  # Print the following depending on the color input
5  # if green
6  #    -> "Go"
7  # elif green
8  #    -> "Wait..."
9  # elif green
10 #    -> "Stop"
```

Else Statement (Last Resort)

The else statement runs a piece of code when every condition fails




```
if condition_1:
    """Code A"""

elif condition_2:
    """Code B"""

elif condition_3:
    """Code C"""

else:
    """Last Resort"""
```

Else Statement

The else statement is often used to notify on unexpected issues in the input

```
1  you_said = input("You said: ")
2
3  if you_said == "Wish":
4      print("107.5")
5  elif you_said == "Hello":
6      print("...it's me")
7  elif you_said == "Jopay":
8      print("...kamusta ka na")
9  elif you_said == "Black Pink":
10     print("...in your area")
11 else:
12     print("I don't know that song!")
```

Else Statement Example

```
1  you_said = input("You said: ")
2
3  if you_said == "Wish":
4      print("107.5")
5  elif you_said == "Hello":
6      print("...it's me")
7  elif you_said == "Jopay":
8      print("...kamusta ka na")
9  elif you_said == "Black Pink":
10     print("...in your area")
11 else:
12     print("I don't know that song!")
```

User enters "Hey"

False

False

False

False

FINAL RESORT

```
I don't know that song!
```

Quick Exercise: Traffic Lights (version 2)

14_traffic_lights.py

```
1  # Ask the user input for a color
2  color_input = input("Please enter a color: ")
3
4  # Print the following depending on the color input
5  # if green
6  #    -> "Go"
7  # elif green
8  #    -> "Wait..."
9  # elif green
10 #    -> "Stop"
   # else
   #    -> "Malfunction"
```

Multiple If's versus If-Elif's

If-elif statements ensure only one option runs. That's not the case for multiple if statements

```
1  grade = 85
2
3  if grade >= 90:
4      print("A")
5  if grade >= 80:
6      print("B")
7  if grade >= 70:
8      print("C")
```

```
B
C
```

```
1  grade = 85
2
3  if grade >= 90:
4      print("A")
5  elif grade >= 80:
6      print("B")
7  elif grade >= 70:
8      print("C")
```

```
B
```

Multiple If's versus If-Elif's

```
1  battery = int(input("Battery percentage: "))
2
3  if battery >= 80:
4      print("Full Battery")
5  if battery >= 40:
6      print("Good Battery")
7  if battery >= 15:
8      print("Low Battery")
9  if battery > 0:
10     print("Critically Low Battery")
11 else:
12     print("No Battery")
```

If-Else Condition Example 01

For most cases, only the if-else statements are used

```
1  age = int(input("Enter age: "))
2  if age >= 18:
3      print("Old enough to watch movie")
4  else:
5      print("Too young to watch movie")
```

If-Else Condition Example 02

```
1  balance = 150
2  price = 200
3
4  if balance >= price:
5      print("Payment successful")
6  else:
7      print("Insufficient funds")
```

Quick Exercise: Password Check (v3)

13_password_check.py

```
1  # Expected password (you can change the value)
2  correct_password = "pass"
3
4  # Enter user password
5  password_input = input("Please provide password: ")
6
7  # If correct_password_give, print: "Access Granted"
8  # If not correct_password, print: "Access Denied"
9  correct_password_given = None
10
11 print("Access Granted")
12 print("Access Denied")
```

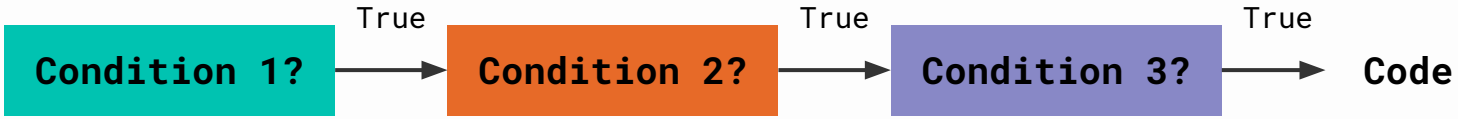
Logical Operators

Simplifying conditionals

And Operator

You can use the **and** operator to make the condition more strict

```
1  has_government_id = True
2  has_nbi_clearance = True
3  has_registered = True
4
5  if has_government_id and has_nbi_clearance and has_registered:
6      print("You can now apply!")
```



And Operator Example

You can use the `and` operator to make the condition more strict

```
1  money = float(input("Enter money: "))
2  stock = int(input("Enter stock: "))
3
4  if money >= 100 and stock > 0:
5      print("You can buy the item!")
6  else:
       print("You can't buy the item")
```

Quick Exercise: Admin Login

15_admin_login.py

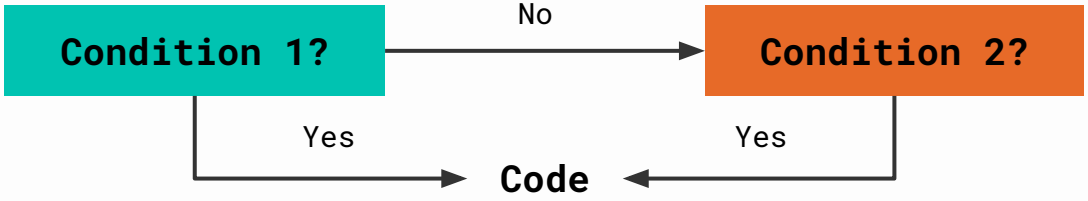
```
1  # Expected username and password(you can change the value)
2  correct_username = "admin"
3  correct_password = "admin"
4
5  # Enter username and password
6  username_input = input("Please provide username: ")
7  password_input = input("Please provide password: ")
8
9  # If username_input equal correct_username
10 #    and password_input equal correct_password
11 #    print: "Access Granted"
12 print("Access Granted")
13 # else, print: "Access Denied"
14 print("Access Denied")
```

Or Operator

Use the **or** operator to add alternative conditions

```
1  response = input("Continue? ")
2  if response == "yes" or response == "YES":
3      print("We will continue!")
```

Make a new file and try this code



Or Operator Example

Use the `or` operator to add alternative conditions

```
1  raining = True
2  cold = True
3  trendy = True
4
5  if raining or cold or trendy:
6      print("Wear a jacket")
```

Quick Exercise: Edit Access

16_edit_access.py

```
1  # Ask the user for their role
2  number = int(input("Enter role: "))
3
4  # If role is "admin" or role is "editor", print the following
5  print("Edit access enabled")
6
7  # Else, print: "Access denied"
8  print("Edit not allowed")
```

Not Operator

A boolean value or statement can be reversed or negated using the not operator

```
1  print(not True)
```

```
2  print(not False)
```

```
3  authenticated = False
4  if not authenticated:
5      print("Access Denied")
```


Consider:

How is this logic represented in code?

Greeting?

Buffet Discount?

Overtime Bonus?

Food Price (Consider Tax)?

For Loops

Controlled repetitions

Defining a List

```
items =
["milk","egg","ice"]
print(items)
```

Quick Exercise: Bookmarks

17_bookmarks.py

```
1  # Define a list of your favorite websites
2  websites = ["facebook.com", "youtube.com", ...]
3
4  # Print the entire list of websites
5  print(websites)
```

For Loop

```
items =
["milk","egg","ice"]
for item in items:
    print(item)
```

For Loop

A for loop goes through the items or elements of a list one at a time by assigning to a variable

```
1  items = ["milk","egg","ice"]
2  for item in items:
3      print(item)
```

"milk"	"egg"	"ice"

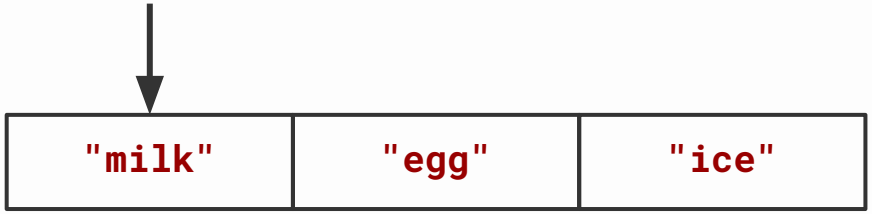
```
1  item = "milk"
2  print(item)
3
4  item = "egg"
5  print(item)
6
7  item = "ice"
8  print(item)
```

For Loop

A for loop goes through the items or elements of a list one at a time by assigning to a variable

```
1  items = ["milk","egg","ice"]
2  for item in items:
3      print(item)
```

item="milk"




```
1  item = "milk"
2  print(item)
3
4  item = "egg"
5  print(item)
6
7  item = "ice"
8  print(item)
```

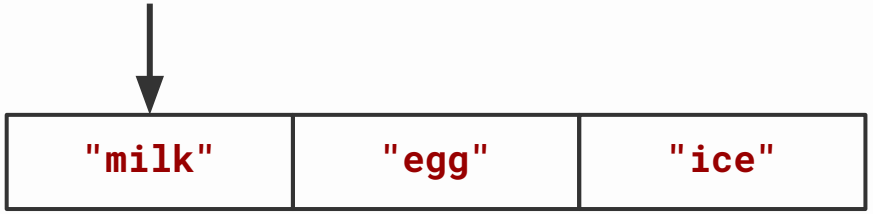
For Loop

A for loop goes through the items or elements of a list one at a time by assigning to a variable

```
1  items = ["milk","egg","ice"]
2  for item in items:
3      print(item)
```

```
milk
```

item="milk"



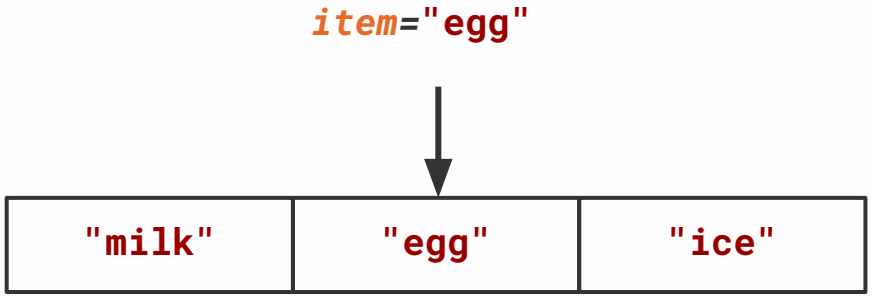

```
1  item = "milk"
2  print(item)
3
4  item = "egg"
5  print(item)
6
7  item = "ice"
8  print(item)
```

For Loop

A for loop goes through the items or elements of a list one at a time by assigning to a variable

```
1  items = ["milk","egg","ice"]
2  for item in items:
3      print(item)
```

```
milk
```



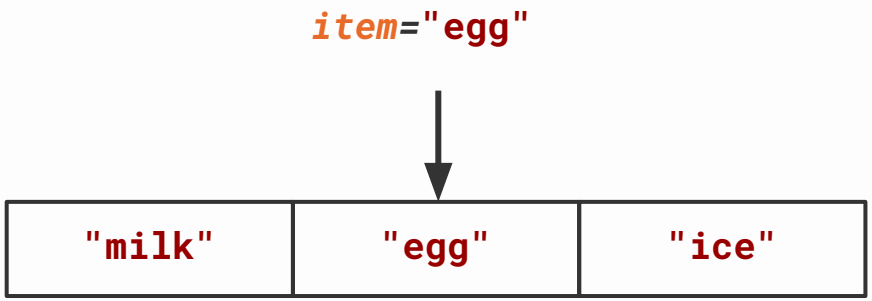

```
1  item = "milk"
2  print(item)
3
4  item = "egg"
5  print(item)
6
7  item = "ice"
8  print(item)
```

For Loop

A for loop goes through the items or elements of a list one at a time by assigning to a variable

```
1  items = ["milk","egg","ice"]
2  for item in items:
3      print(item)
```

```
milk
egg
```



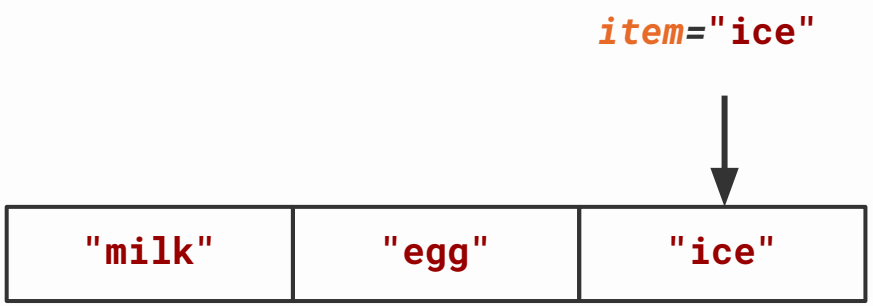

```
1  item = "milk"
2  print(item)
3
4  item = "egg"
5  print(item)
6
7  item = "ice"
8  print(item)
```

For Loop

A for loop goes through the items or elements of a list one at a time by assigning to a variable

```
1  items = ["milk","egg","ice"]
2  for item in items:
3      print(item)
```

```
milk
egg
```




```
1  item = "milk"
2  print(item)
3
4  item = "egg"
5  print(item)
6
7  item = "ice"
8  print(item)
```

For Loop

A for loop goes through the items or elements of a list one at a time by assigning to a variable

```
1  items = ["milk","egg","ice"]
2  for item in items:
3      print(item)
```

```
milk
egg
ice
```


```
1  item = "milk"
2  print(item)
3
4  item = "egg"
5  print(item)
6
7  item = "ice"
8  print(item)
```

For Loop Example 01: Prints

For prints are often used to print values one at a time

```
1  notifications = ["Battery low", "New message", "New Update"]
2
3  for notification in notifications:
4      print("Alert:", notifications)
```

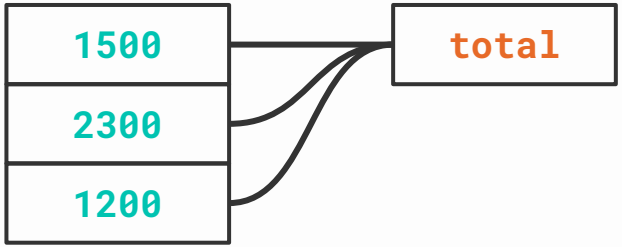
```
Battery low
New message
Update available
```

For Loop Example 02: Aggregation

A common task in for loops is combining all of the items into one value

```
1  expenses = [1500, 2300, 1200]
2  total = 0
3
4  for amount in expenses:
5      total += amount
6
7  print("Total expenses:", total)
```

```
Total expenses: 5000
```

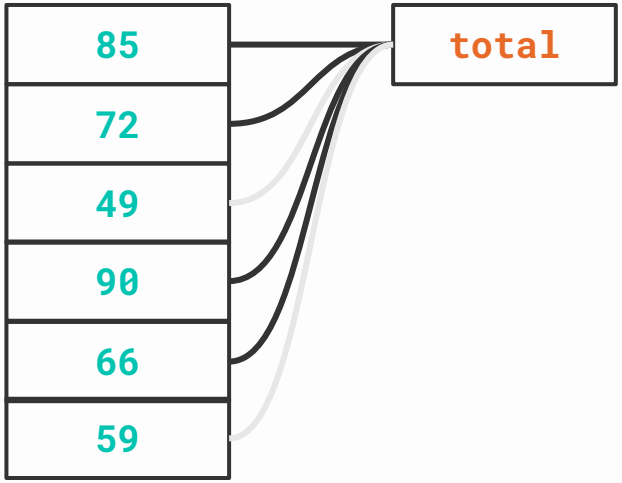


For Loop Example 03: Counting

Finally, another common task besides aggregation is counting

```
1  grades = [85, 72, 49, 90, 66, 59]
2  passing = 0
3
4  for grade in grades:
5      if grade >= 60:
6          passing += 1
7
8  print("Passing", passing)
```

```
Passing: 4
```



Quick Exercise: Bookmarks (version 2)

17_bookmarks.py

```
1  # Define a list of your favorite websites (add or change below)
2  websites = ["facebook.com", "youtube.com"]
3
4  # Print the entire list of websites (one at a time)
5  print(websites)
```

Fixed Repetition

Using a *range(n)* function instead of a list makes the code repeat that many times

```
1  for item in range(3):
2      print("This will be repeated")
```

```
This will be repeated
This will be repeated
This will be repeated
```

Quick Exercise: Repetition

18_repetition.py

```
1  # Long Message
2  message = "This is a very long message that's hard to type"
3
4  # Print the message eleven times
5  print(message)
```

For Range Loop

The *range(n)* function actually generates a list from *0* to *n-1*

```
1 for item in range(3):
2     print(item)
```

```
0
1
2
```

```
1 numbers = [0, 1, 2]
2 for item in numbers:
3     print(item)
```

Quick Exercise: Counting

18_counting.py

```
1  # Ask the user for a number
2  limit = int(input("Enter number: "))
3
4  # Print the numbers 0 to limit
5  print()
```

Range() with different start

The *range(start,end)* is a variation of *range(n)* function that generates a list from *start* to *end-1*

```
1 for item in range(1, 6):
2     print(item)
```

```
1
2
3
4
5
```

Quick Exercise: Counting (version 2)

18_counting.py

```
1  # Ask the user for a starting and ending number
2  start = int(input("Enter start: "))
3  end = int(input("Enter end: "))
4
5  # Print the numbers start to end
6  print()
```

Range() with different step

The *range(start,end,step)* is a variation of *range(n)* function that generates a list from *start* to *end-1* and skips count by *step*

```
1 for item in range(2, 11, 2):
2     print(item)
```

```
2
4
6
8
10
```

Quick Exercise: Tens

19_tens.py

```
1  # Print the following pattern up to 100
2  # 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
3
4  print()
```

While Loops

Dynamic repetitions

Dynamic Repetition

Some repetitions cannot be fixed or determined

```
1  correct_password = "pass"
2
3  password = input("Enter password: ")
```

Dynamic Repetition

Some repetitions cannot be fixed or determined

```
1  correct_password = "pass"
2
3  password = input("Enter password: ")
4  if password != correct_password:
5      password = input("Enter password: ")
```

Dynamic Repetition

Some repetitions cannot be fixed or determined

```
1  correct_password = "pass"
2
3  password = input("Enter password: ")
4  if password != correct_password:
5      password = input("Enter password: ")
6
7      if password != correct_password:
8          password = input("Enter password: ")
```

Dynamic Repetition

Some repetitions cannot be fixed or determined

```
1  correct_password = "pass"
2
3  password = input("Enter password: ")
4  if password != correct_password:
5      password = input("Enter password: ")
6
7      if password != correct_password:
8          password = input("Enter password: ")
9
10         if password != correct_password:
11             password = input("Enter password: ")
12             ...
```

Repeat until the user gets it right

```
password = input()
while incorrect password :
    password = input()
```

While Loop Example 01

While loops are used when the end of a loop relies on a condition

```
1  correct_password = "pass"
2
3  password = input("Enter password: ")
4  while password != correct_password:
5      password = input("Enter password: ")
```

```
1  correct_password = "pass"
2
3  password = ""
4  while password != correct_password:
5      password = input("Enter password: ")
```

While Loop Example 02

This structure is commonly used to repeat certain tasks until user says otherwise

```
1  stop_program = False
2
3  while not stop_program:
4      choice = input("Provide command: ")
5      if choice == "command 1":
6          print("command 1 done")
7      elif choice == "command 2":
8          print("command 2 done")
9      elif choice == "command 3":
10         print("command 3 done")
11     elif choice == "exit":
12         stop_program = True
```

Quick Exercise: Console Counter

20_console_counter.py

```
1  count = 0
2  stop_program = False
3  while not stop_program:
4      choice = input("Provide command: ")
5
6      # If choice equals add, increase count
7
8      # elif choice equals sub, decrease count
9
10     # elif choice equals double, double count
11
12     elif choice == "exit":
13         stop_program = True
```

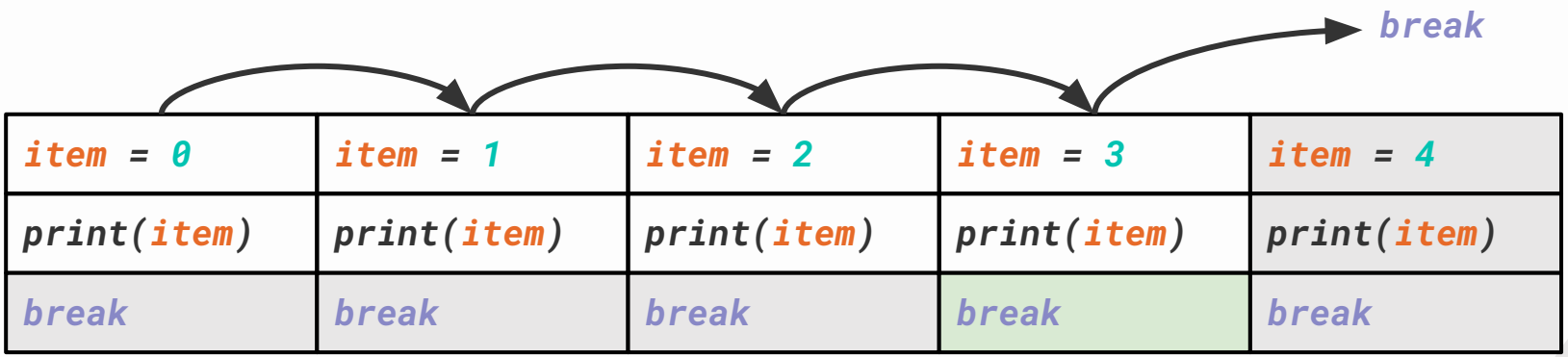
Loop Breaks

Exit the common mold

Break Keyword

The **break** keyword immediately stops the loop

```
1  for item in range(100):
2      print(item)
3      if item == 3:
4          break
```



While Loop: Early Break

The **break** keyword immediately stops the loop

```
1  max_attempt = 3
2  correct_password = "pass"
3
4  for attempt in range(max_attempt):
5      password = input("Enter password: ")
6      if password == correct_password:
7          print("Access granted")
8          break
9      else:
10         attempts += 1
11
12 if attempt == max_attempt:
13     print("Account locked")
```

Quick Exercise: Search

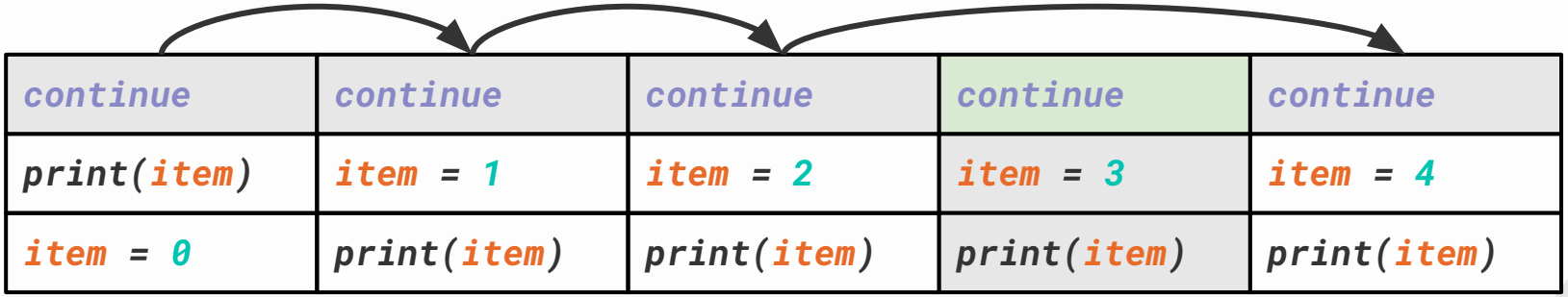
21_search.py

```
1  items = ["rice", "noodles", "toyo", "spam", "coffee"]
2  item_to_find = "spam"
3
4  for item in items:
5      """If item equals the item_to_find, print and exit loop"""
6
7
```

Continue Keyword

The **continue** keyword skips the succeeding code

```
1  for item in range(100):
2      if item == 3:
3          continue
4      print(item)
```



Quick Exercise: Skip Range

22_skip_range.py

```
1  for item in range(100):
2      # Change code to skip printing numbers 20 to 80.
3      print(item)
4
```

H2

Cost Calculator v2

Make the previous version more dynamic!

Cost Calculator (version 2)

10_cost_calculator.py

```
1  # Ask the user how many items to purchase
2  item_count = int(input("Enter item count: "))
3  total = 0
4
5  # For every item in range of item_count, ask for an item price
6  item_price = int(input("Enter item count: "))
7
8  # Add the item price to the total price
9
10
11 print(total)
```

Item Quantity?

04

Functions

First step to code organization

Function Basics

Creating your own subprocesses

Sum Calculator

```
1  numbers = [1, 3, 5, 7, 2, 4, 6]
2  total = 0
3  for number in numbers:
4      total += number
5
6  print(total)
```

```
7  new_numbers = [9, 3, 0, 1, 2, 7]
```

Sum Calculator

```
1  numbers = [1, 3, 5, 7, 2, 4, 6]
2  total = 0
3  for number in numbers:
4      total += number
5
6  print(total)
```

```
7  new_numbers = [9, 3, 0, 1, 2, 7]
8  total_2 = 0
9  for number in new_numbers:
10     total_2 += number
11
12 print(total_2)
```

What if I need to calculate another list?

Sum Calculator

```
1  numbers = [1, 3, 5, 7, 2, 4, 6]
2  total = sum(numbers)
3  print(total)
```

```
4  new_numbers = [9, 3, 0, 1, 2, 7]
5  total_2 = sum(new_numbers)
6  print(total_2)
```

Sum Calculator

```
1  numbers = [1, 3, 5, 7, 2, 4, 6]
2  total = sum(numbers)
3  print(total)
```

```
4  new_numbers = [9, 3, 0, 1, 2, 7]
5  total_2 = sum(new_numbers)
6  print(total_2)
```

```
4  final_numbers = [1, 2, 3]
5  total_3 = sum(final_numbers)
6  print(total_3)
```

No need to copy paste code

Simple Function Declaration

The default syntax of a function is shown below:

```
def function_name():
    """processes here"""
```

```
1  def greet():
2      print("Hello, good day to you!")
```

```
3  greet()
```

Regular Code Flow

Python code runs line by line from top to bottom

```
1  print("First Line")
2  print("Second Line")
3  print("Third Line")
```

Function Copy-Pasting

When you have a function, it goes back like it's copy pasting the code in-between

```
1  def extra():
2      print("Extra Line 1")
3      print("Extra Line 2")
```

```
4  print("First Line")
5  extra()
6  print("Second Line")
```

```
1  def extra():
2      print("First Line")
3
4      print("Extra Line 1")
5      print("Extra Line 2")
6
7      print("Second Line")
```

Quick Exercise: Line Generator

23_line_generator.py

```
"""
Create a function line_generator that prints the following:
    Line 1
    Line 2
    Line 3
"""

# Use the function once
line_generator()

```

Simple Input Declaration

The default syntax of a function with a single input has the following syntax:

```
def function_name(variable_name):
    """processes here"""
```

```
1 def greet(username):
2     print(f"Hello {username}, good day to you!")
```

```
3 greet("Joseph")
```

Quick Exercise: Line Generator (version 2)

23_line_generator.py

```
"""
Create a function line_generator that has a parameter number
and prints the following:
    Line 1
    Line 2
    ...
    Line number
"""

# Use the function once (pick any number input)
line_generator(4)
```

Multiple Input Declaration

Using more than one input means requires commas

```
def function_name(variable_name_1, variable_name_2):
    """processes here"""
```

```
1 def greet(username, message):
2     print(f"Hello {username}, {message}")
```

```
3 greet("Joseph", "Nice to meet you!")
```

Quick Exercise: Line Generator (version 3)

23_line_generator.py

```
"""
Create a function line_generator
that has a parameter number and parameter message
and prints the following:
    message 1
    message 2
    ...
    message number
"""

# Use the function once (pick any number input and message)
line_generator(4, "Line")
```

Optional Parameter

You can use a default value for the function inputs

```
def function_name(variable_name_1, variable_name_2=default):
    """processes here"""
```

```
1 def greet(username, message="Nice to meet you!"):
2     print(f"Hello {username}, {message}")
```

```
3 greet("Joseph")
```

Optional Parameter (Overriding)

You can use a default value for the function inputs

```
def function_name(variable_name_1, variable_name_2=default):
    # processes here
```

```
1 def greet(username, message="Nice to meet you!"):
2     print(f"Hello {username}, {message}")
```

```
3 greet(username="Joseph", message="Hajimemashite!")
```

Quick Exercise: Line Generator (final)

23_line_generator.py

```
"""
Create a function line_generator
that has a parameter number and parameter message
with default values 3 and "Line"
and prints the following:
    message 1
    message 2
    ...
    message number
"""

# Use the function once with all defaults
line_generator()
```

Function Returns

Simplifying calculations and data handling

Return Value

Functions return values that can be put in a variable

```
def function_name(...):
    # Processes here
    return output
```

```
1  def add(num1, num2):
2      result = num1 + num2
3      return result
4
5  add_result = add(1, 2)
6  print(add_result)
```

Return Function Example 01

Return functions are used to provide context to a calculation

```
1  def to_fahrenheit(celsius):
2      return celsius * 9/5 + 32
3
4  print(to_fahrenheit(30))
```

Return Function Example 02

Return functions also make common operations with complicated formulas easier to use

```
1  def distance(x, y):
2      return (x**2 + y**2)**(1/2)
```

```
4  first_distance = distance(3, 10)
5  second_distance = distance(10, 5)
```

Return Function Example 03

Functions are also used to augment strings

```
1  def happy(string):
2      return string + " :D"
3
4  message = "Hello World"
5  happy_message = happy(message)
6  print(happy_message)
```

Return versus Print

The return keyword does not print the value in the console

```
1  def add(num1, num2):
2      result = num1 + num2
3      return result
4
5  add(1, 2)
```

```
1  def add(num1, num2):
2      result = num1 + num2
3      print(result)
4
5  add(1, 2)
```

```
3
```

Return versus Print

The return keyword allows you to store the value in a variable instead

```
1  def add(num1, num2):
2      result = num1 + num2
3      return result
4
5  add_result = add(1, 2)
6  print(add_result)
```

3

```
1  def add(num1, num2):
2      result = num1 + num2
3      print(result)
4
5  add_result = add(1, 2)
6  print(add_result)
```

None

Return is Final!

When you return in a function it skips everything else after it!

```
1  def add(num1, num2):
2      result = num1 + num2
3      return result
4      print(f"The result is: {result}")
```

skipped

```
5  result = add(3, 4)
6  print(result)
```

Quick Exercise: Number Doubler

24_number_doubler.py

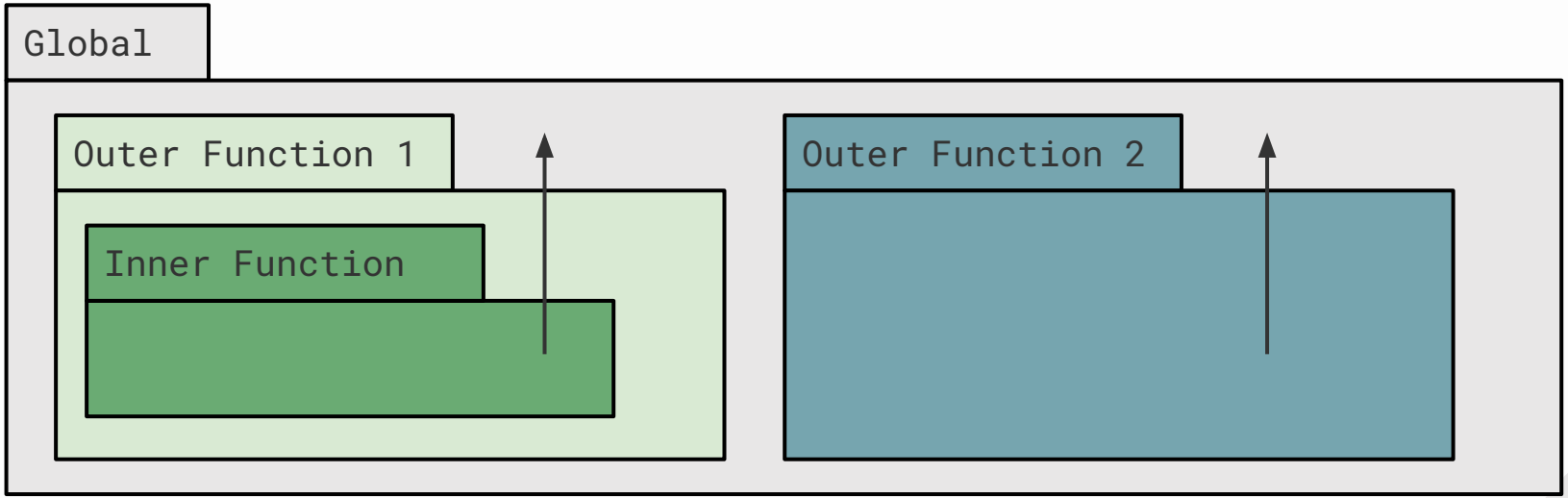
```
"""
Create a function named double
that takes a parameter number
and return twice the number

example:
     double(3) = 6
     double(20) = 40
"""

# Use the function once
x = 3
result = double(x)
print(result)
```

Function Scope

Determining variable lifetime

Function Scoping

The general rule for variable scope is that the variable name is searched starting from the innermost to the outermost



Functions can read outside

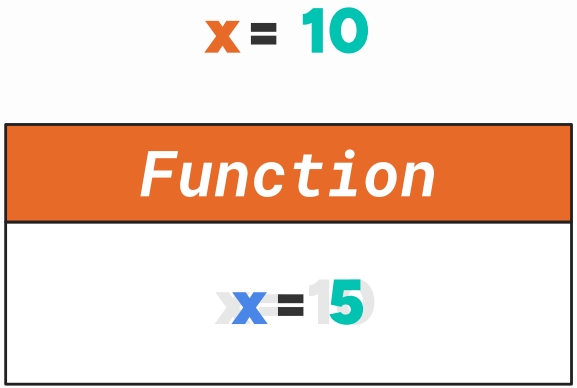
Function can detect and print variables outside of it

```
x = 10
def function():
    print("Inner", x)

print("Outer", x)
function()
print("Outer", x)
```

But functions can’t write outside

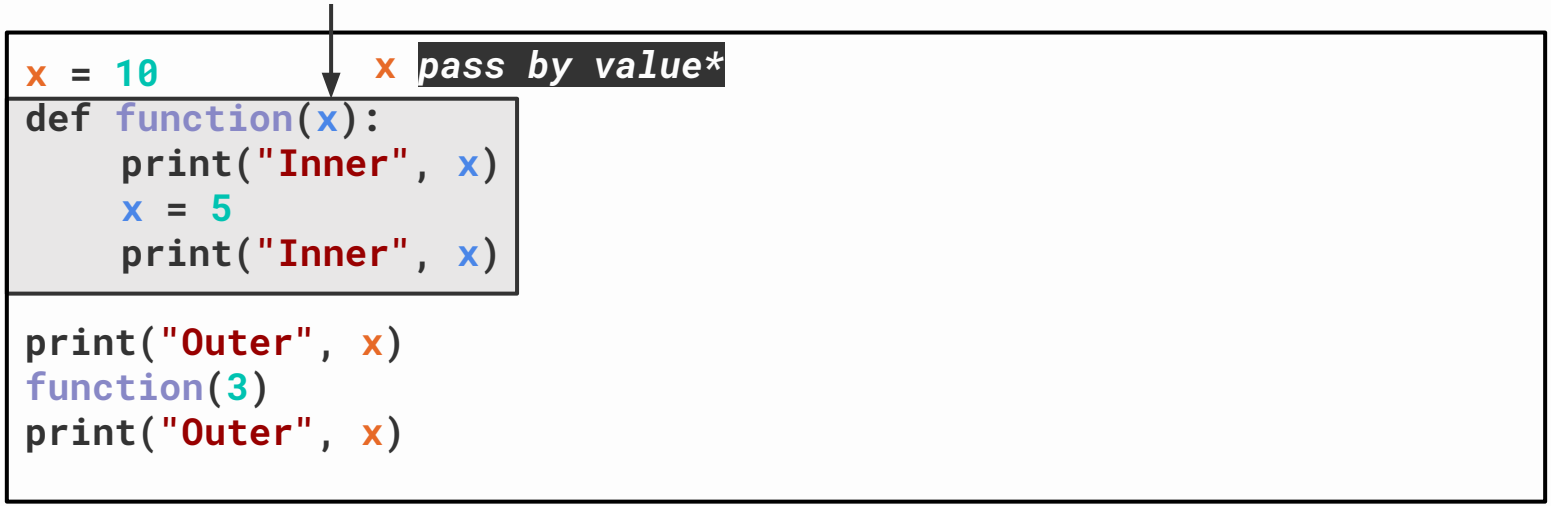
Functions can’t change variables outside because this is making another variable with the same name as the one outside

```
x = 10
def function():
    x = 5
    print("Inner", x)

print("Outer", x)
function()
print("Outer", x)
```



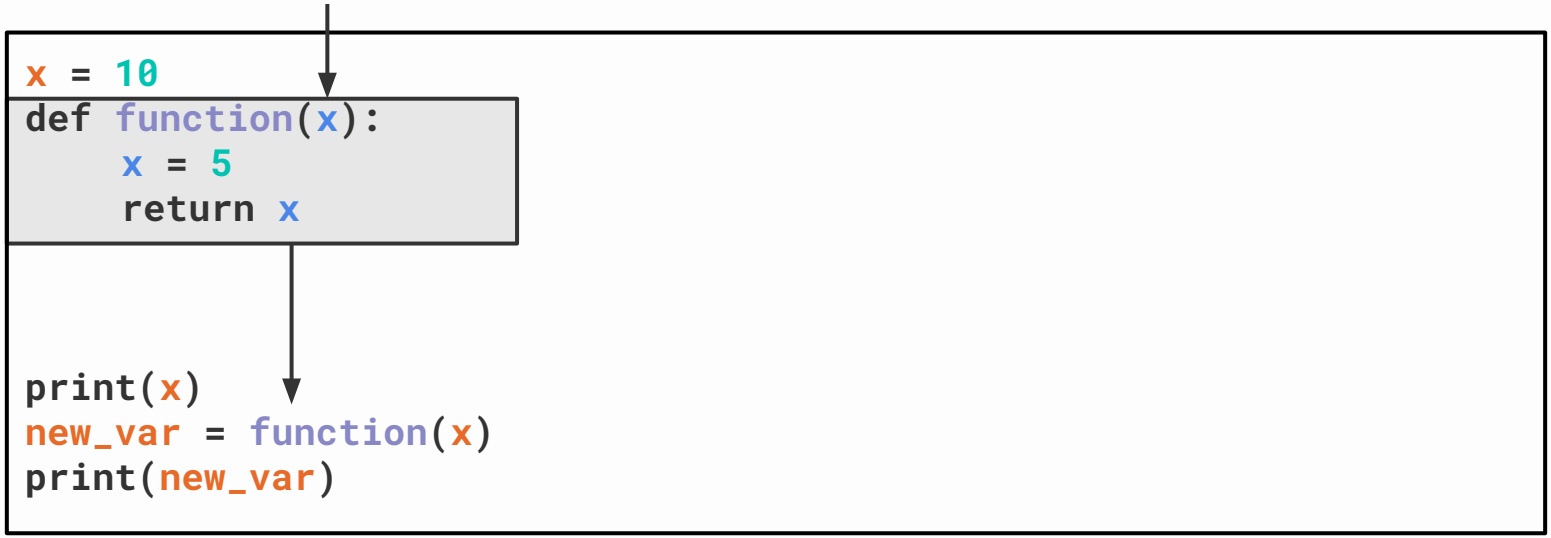
But functions can’t write outside

Even if the variable is given as an input, this does not change anything

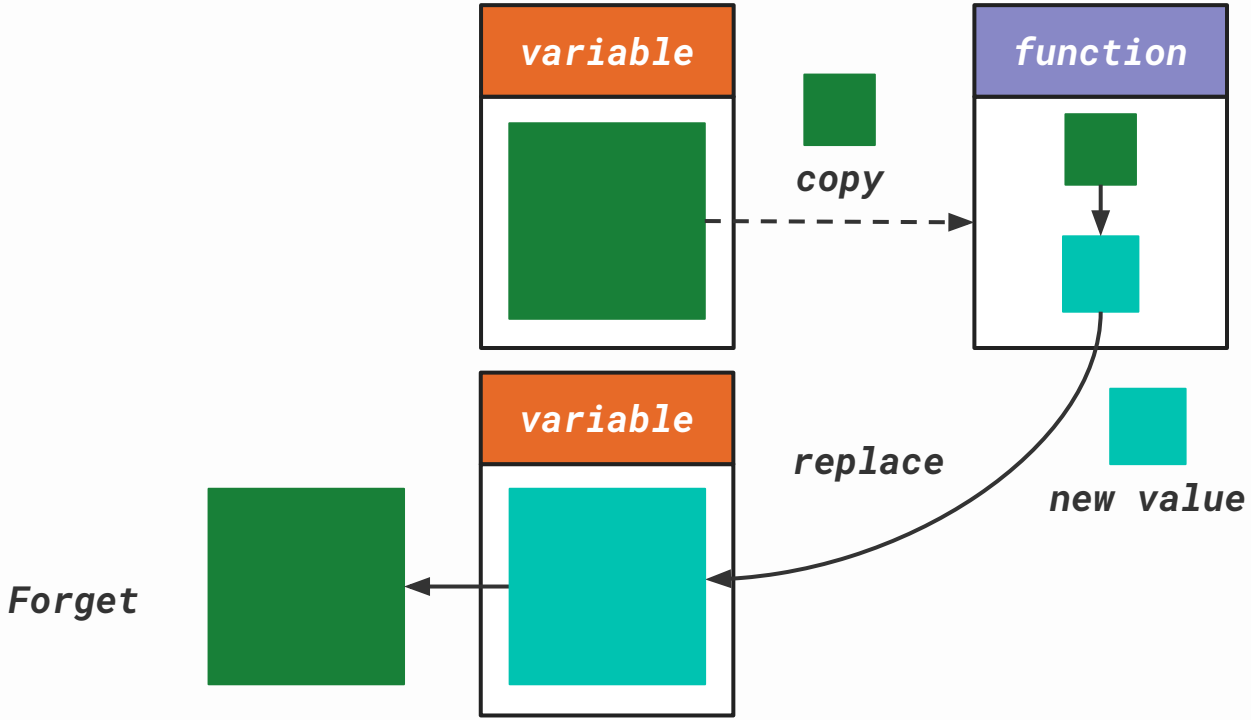


Overwrite using Return

Even if the variable is given as an input, this does not change anything



new_variable = function(variable)



Full Process Example

```
1  def get_expenses():
2      food = float(input("Food expense: "))
3      transpo = float(input("Transport expense: "))
4      return food + transpo
5
6  def get_budget():
7      return float(input("Enter your budget: "))
8
9  def check_budget(total, budget):
10     if total > budget:
11         return "Over budget!"
12     return "Within budget."
13
14 total = get_expenses()
15 budget = get_budget()
16 print("Status:", check_budget(total, budget))
```

H3

Coffee Price

Simplify regular calculations with a function

Coffee Price

25_coffee_price.py

```python
def coffee_price(kind, size, has_discount):
"""
Returns the final price with the following scheme:
    Base Price: Americano (100), Latte (110), Cappuccino (120)
    Size Multiplier: Small (x0.8), Medium  (x1.0), Large (x1.2)
    Has Discount: x0.88 (removed VAT)
"""
    return 0

print(coffee_price("Latte", "Large", True))
print(coffee_price("Americano", "Small", False))
```

05

Error Handling

Making the code secure by preparing for errors

Possible Errors

Simple mistakes or erratic user input can cause errors in the code

```
print(5 / 0)
```

Exception Catching

To prevent complete stops on an exception, use the `try-catch` statements

```
1  try:
2      print(5 / 0)
3  except:
4      print("Please don't divide by zero")
```

Specific Exceptions

You can catch the specific error by adding the name to the right of the except keyword.

```
1  try:
2      print(5 / 0)
3  except ZeroDivisionError:
4      print("Please don't divide by zero")
```

Error Examples	Description
`TypeError()`	Operation applied to data with the wrong type
`ValueError()`	Function or operation got an inappropriate value
`ZeroDivisionError()`	Specifically occurs when dividing by zero

Quick Exercise: Number Error

26_number_error.py

```
1  # Fix possible error if the input given is not a number
2  number_input = int(input("Number: "))
3  print(number_input)
```

Multiple Exceptions

The try-except statements can anticipate multiple errors. Checking is by order of listing.

```
1  try:
2      user_input = int(input("Enter Number: "))
3      result = 5 / user_input
4  except ValueError:
5      print("Input is not a valid number")
6  except ZeroDivisionError:
7      print("Number is a zero!")
8  except KeyboardInterrupt:
9      print("You stopped the code prematurely!")
```

Error Raising

You can trigger errors using the `raise` keyword, followed by the error name and parentheses

```
raise Exception()
```

```
raise ValueError()
```

```
raise ValueError("Custom message here")
```

Error Raising Example

```
1  try:
2      user_input = int(input("Enter Number: "))
3      if user_input < 0:
4          raise ValueError()
5
6  except ValueError:
7      print("We don't accept strings or negatives!")
```

Final Code Execution

Given a line of code that has to run whether the code failed or not...

```
try:
    print(5 / 0)
except:
    print("Please don't divide by zero")
```

Full Exception Handling

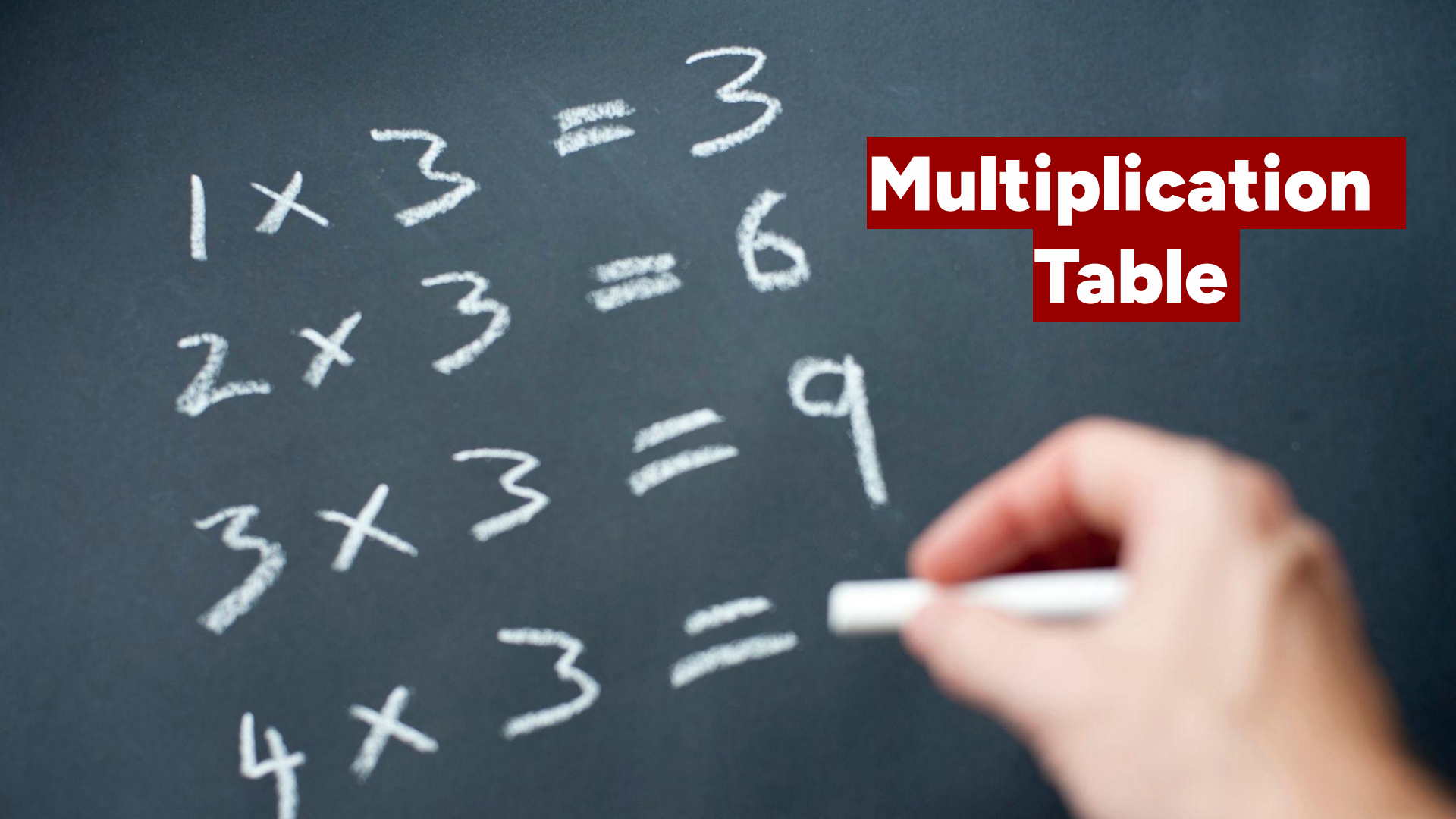
The finally keyword can be used to ensure a line of code runs no matter what happens

```
1  try:
2      print(5 / 0)
3  except:
4      print("Please don't divide by zero")
5  finally:
6      print("Code completed!")
```

06

Lab Session

Overview of the Course and Python in General


1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 =
Multiplication Table

Multiplication Table

Ask the user for an integer input

```
1 number = int(input("Pick a number: "))
```

Print the multiplication table for that `number`

```
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
…
3 x 10 = 30
```

Challenge: Robust Input

Challenge: Alignment

Quick Draw

Prerequisite: Random Choice

In case we need to simulate randomness. First, put this at the top of your code.

```
1 from random import choice
```

This allows us to use the given function that returns a random item from a list

```
2 options = ["rock", "paper", "scissors"]
3 random_option = choice(options)
4 print(random_option )
```

Recommended Project: Quick Draw

Ask the user for an input

```
1 user_choice = input("Pick a choice (rock/paper/scissors): ")
```

Make a random choice for the computer

```
2 cpu_choice = ...
```

Depending on their choices, tell if the user won, lost, or there was a draw:

```
You win!
```

```
You Lost!
```

```
Draw!
```

Challenge: Robust Input

Challenge: Multi-rounds


Positive Input

Safe Integer

Ask the user for an input that should be a number

```
1 number = input("Enter number: ")
```

Then try to convert this into an integer using the following:

```
2 number_converted = int(number)
```

But this will cause an error if the user gives a non-integer input. Can you make the error print the following if the input is invalid?

```
Invalid input. Please provide a valid integer
```

Positive Integer

Next, the input should be a POSITIVE integer (greater or equal to zero). If the input is not, print this message instead:

```
Invalid input. Please provide a positive integer
```

Continuous Positive Integer

Finally, keep asking the user for input for as long as they do not give a valid, positive, integer

```
Enter number: "Invalid Input 1"
Enter number: "Invalid Input 2"
...
```

Sneak Peak

01

Lists & Tuple

Ordered Group

02

Dictionary & Set

Unordered Group

03

String

Handling Text

04

File Handling

Outside-Code Storage

05

Comprehension

Iteration Shortcut

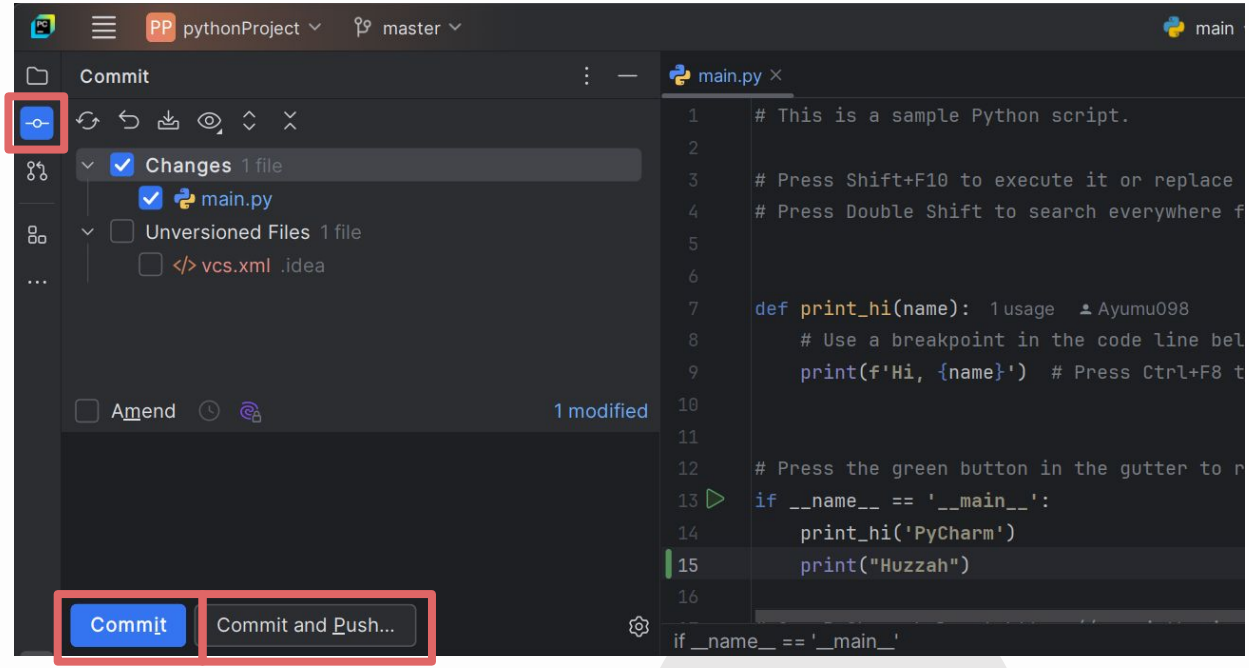
06

Lab Session

Culminating Exercise

Saving Work: Commit and Push

Select **Commit** to save locally only and **Commit and Push** to save remote as well.



Python: Day 01

Introduction to Python and Basic Syntax