Session 4: More Python for API Calls and API Workshop 🛠️

More practice and next steps

📽️ [Link to Presentation]()

Slide 1: Title

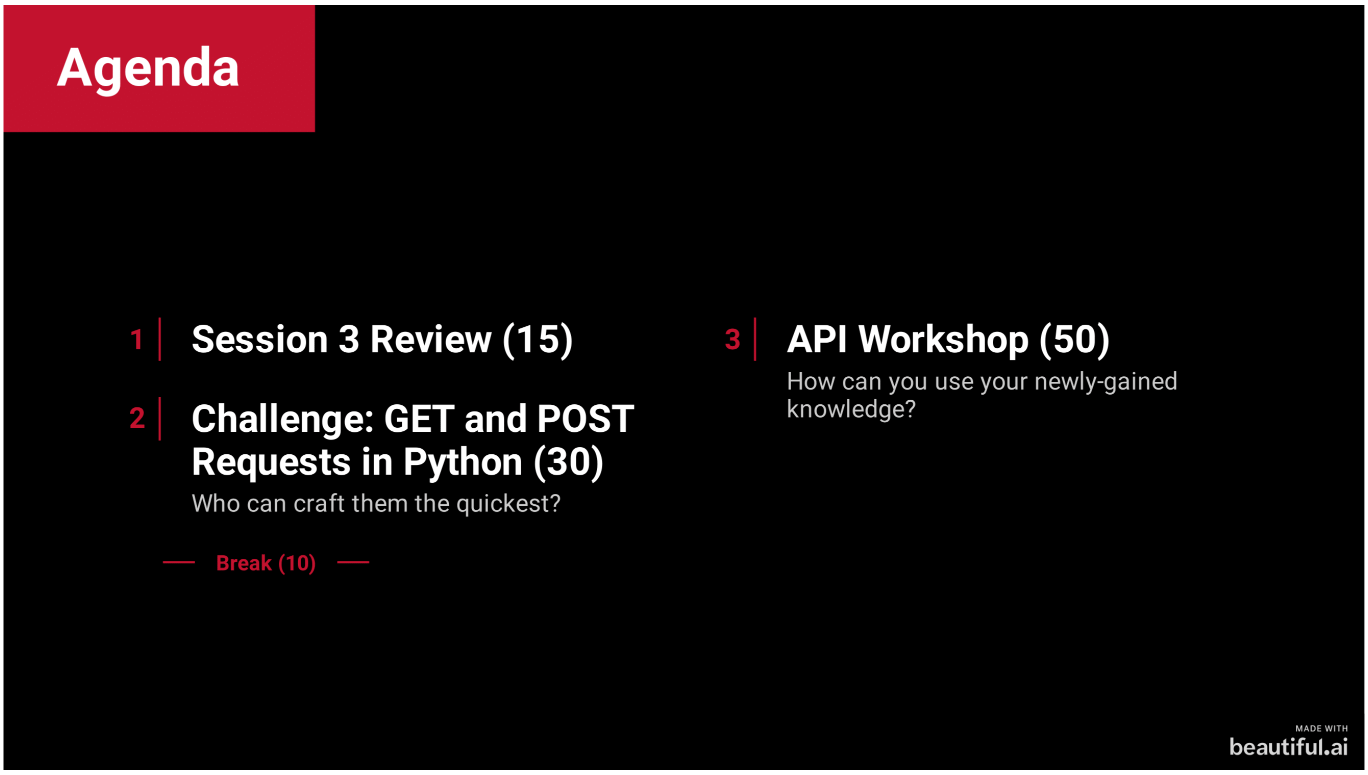
API ALCHEMY: SESSION 4

More Python for API Calls and Workshop

MADE WITH
beautiful.ai

Slide 2: Agenda

Our plan for the day



Agenda

1 Session 3 Review (15)

2 Challenge: GET and POST Requests in Python (30)
Who can craft them the quickest?

Break (10)

3 API Workshop (50)
How can you use your newly-gained knowledge?

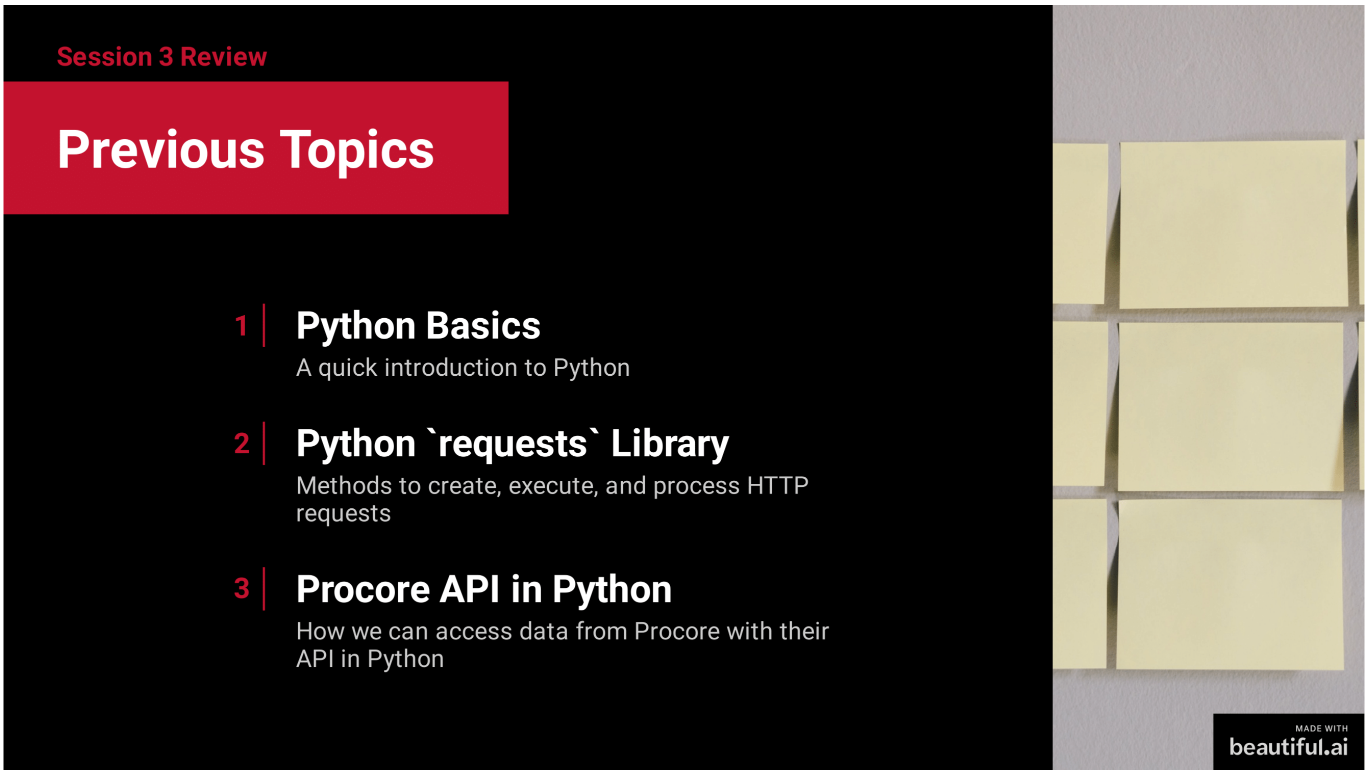
MADE WITH
beautiful.ai

1. Session 3 Review
2. Two Challenges with The Cat API
    1. GET a Random Cat Image
    2. POST a like to the Chicken Nugget's image
3. API Workshop

🧑‍🏫 Session 3 Review

▶ Click to Expand

Slide 4: Previous Topics



Session 3 Review

Previous Topics

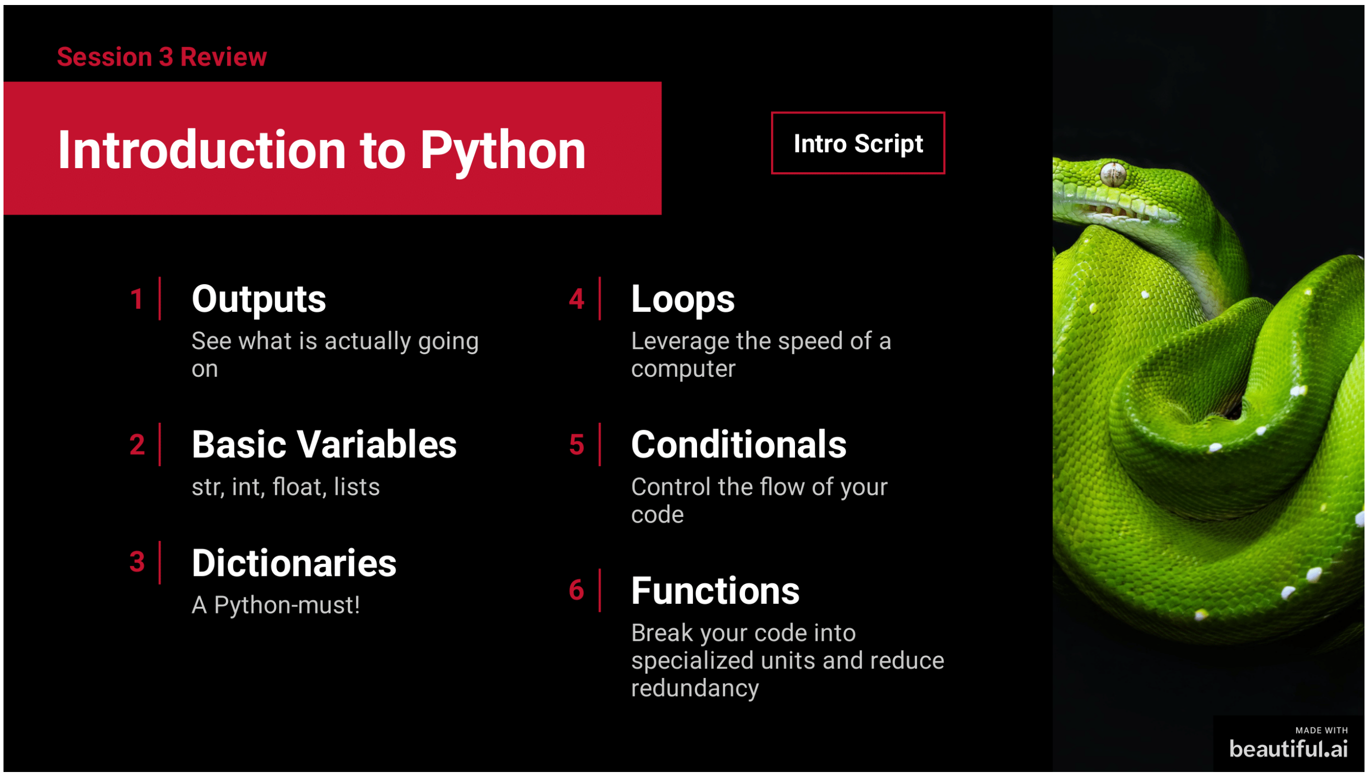
1 **Python Basics**
A quick introduction to Python

2 **Python `requests` Library**
Methods to create, execute, and process HTTP requests

3 **Procore API in Python**
How we can access data from Procore with their API in Python

1. **Python Basics**: A quick introduction to Python
2. **Python `requests` Library**: Methods to create, execute, and process HTTP requests
3. **Procore API in PYthon**: How we can access data from Procore with their API in Python

Slide 5: Introduction to Python



Session 3 Review

Introduction to Python

Intro Script

1 **Outputs**
See what is actually going on

2 **Basic Variables**
str, int, float, lists

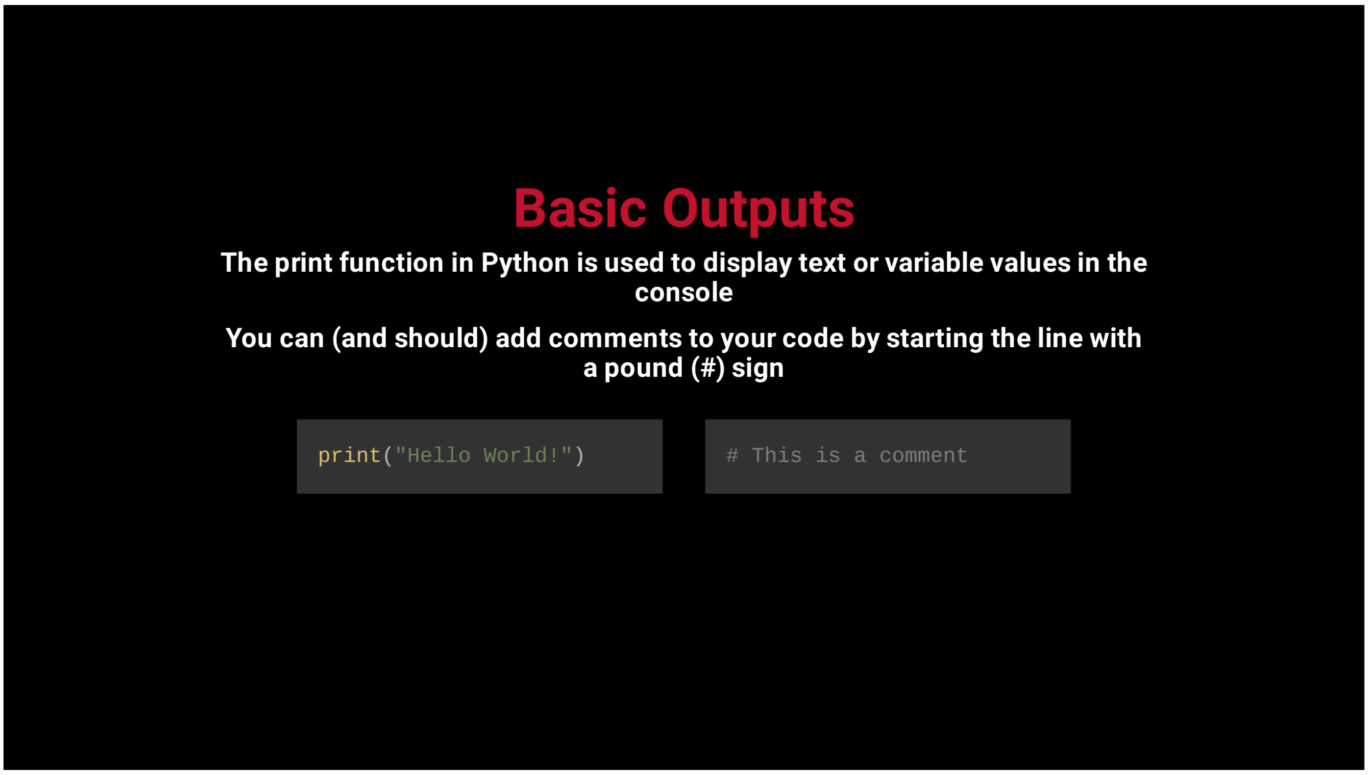
3 **Dictionaries**
A Python-must!

4 **Loops**
Leverage the speed of a computer

5 **Conditionals**
Control the flow of your code

6 **Functions**
Break your code into specialized units and reduce redundancy

Slide 7: Basic Outputs



Basic Outputs

The print function in Python is used to display text or variable values in the console

You can (and should) add comments to your code by starting the line with a pound (#) sign

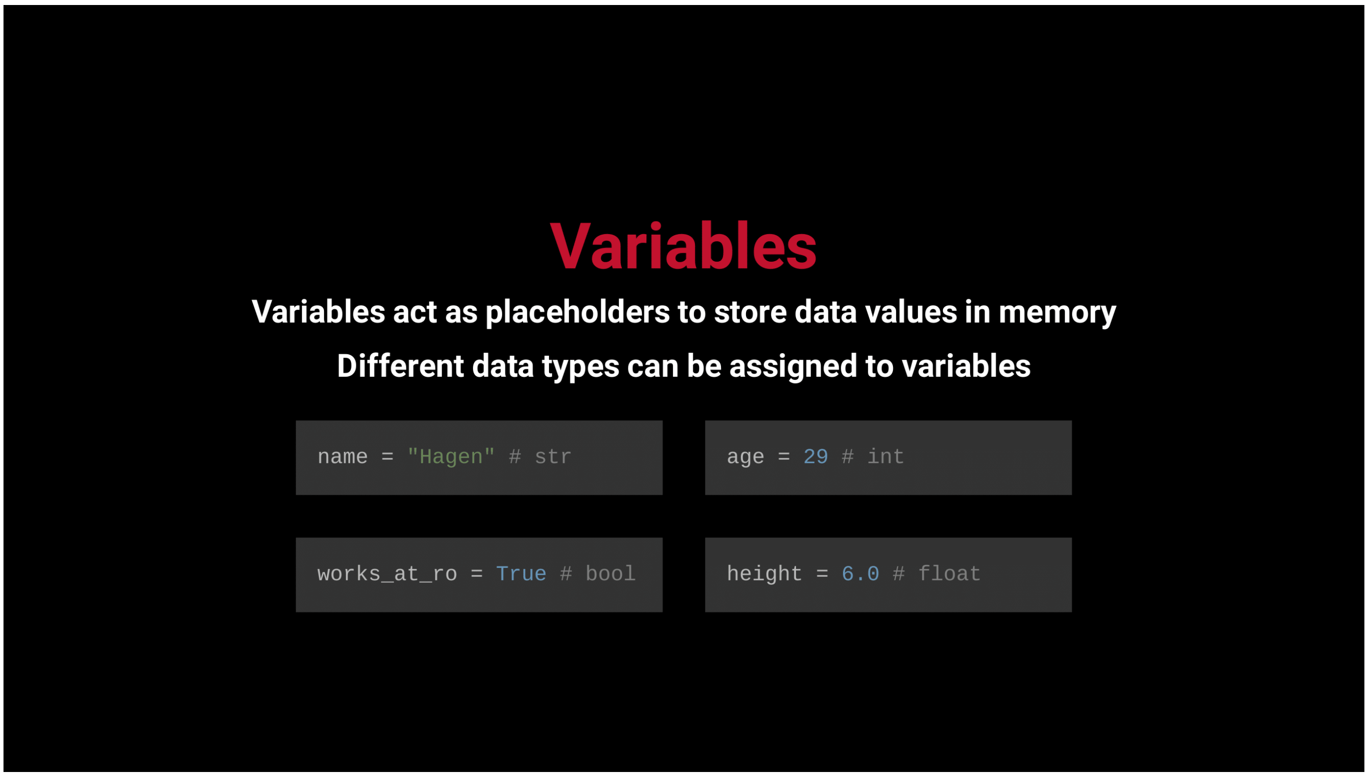
```
print("Hello World!")
```

```
# This is a comment
```

The print function in Python is used to display text or variable values in the console.

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

Slide 9: Variables



Variables

Variables act as placeholders to store data values in memory

Different data types can be assigned to variables

```
name = "Hagen" # str
```

```
age = 29 # int
```

```
works_at_ro = True # bool
```

```
height = 6.0 # float
```

Variables Types

Variables act as placeholders to store data values in memory. Different data types can be assigned to variables such as:

- String: Textual data

```
name = "Hagen"
```

- Integer: Whole numbers

```
age = 29
```

- Float: Decimal numbers

```
height = 6.0
```

- Boolean: binary true or false (0 or 1)

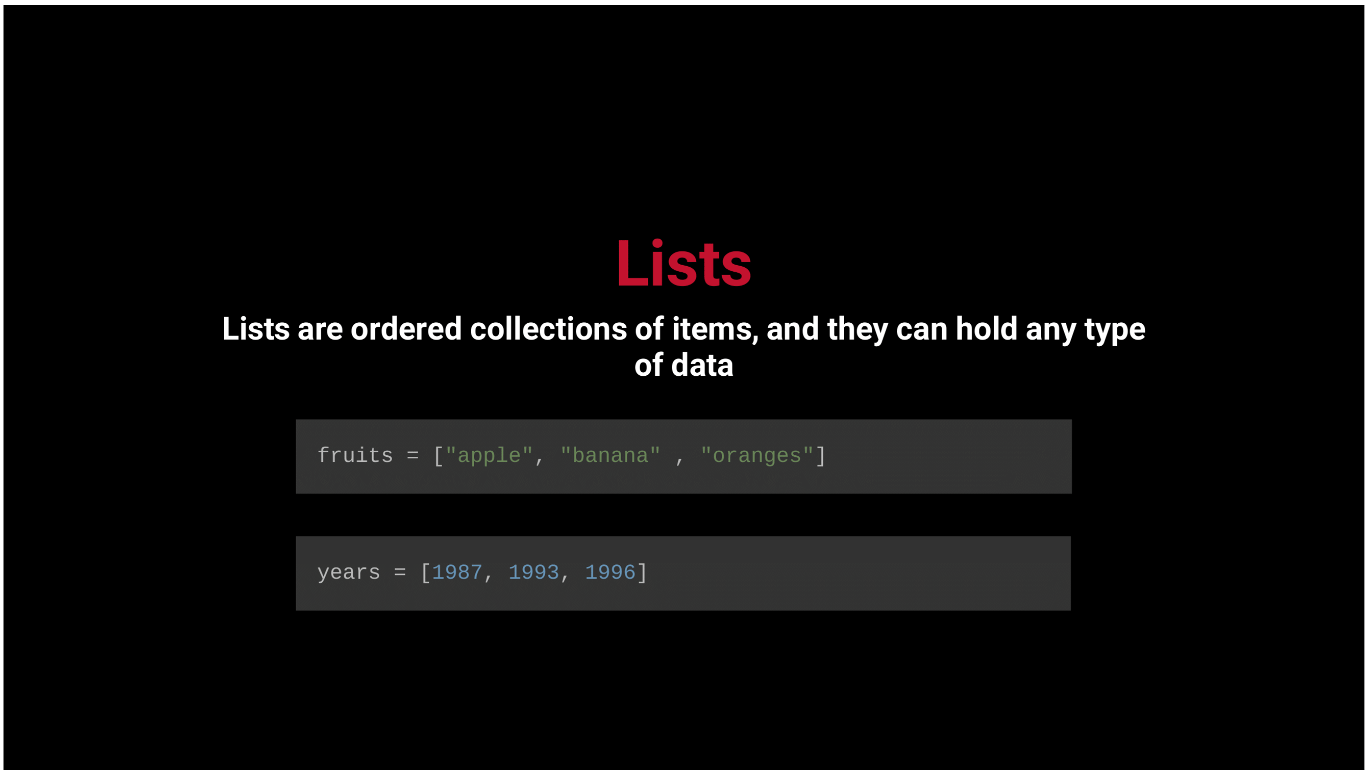
```
works_at_ro = True
```

> Unlike other languages, Python does not require you to declare the type of variable when declaring it. Python will figure out the variable type for you when you run your code.

Variable Naming

- **Starting Character**: Variable names must start with a letter (a-z, A-Z) or an underscore (_). The rest of the name can contain letters, numbers, or underscores.
- **Case-Sensitive**: Variable names are case-sensitive (age, Age, and AGE are three different variables).
- **Reserved Words**: Python has defined keywords like `if`, `else`, `while`, etc.) that cannot be used as variable names.

Slide 10: Lists



Lists

Lists are ordered collections of items, and they can hold any type of data

```
fruits = ["apple", "banana" , "oranges"]
```

```
years = [1987, 1993, 1996]
```

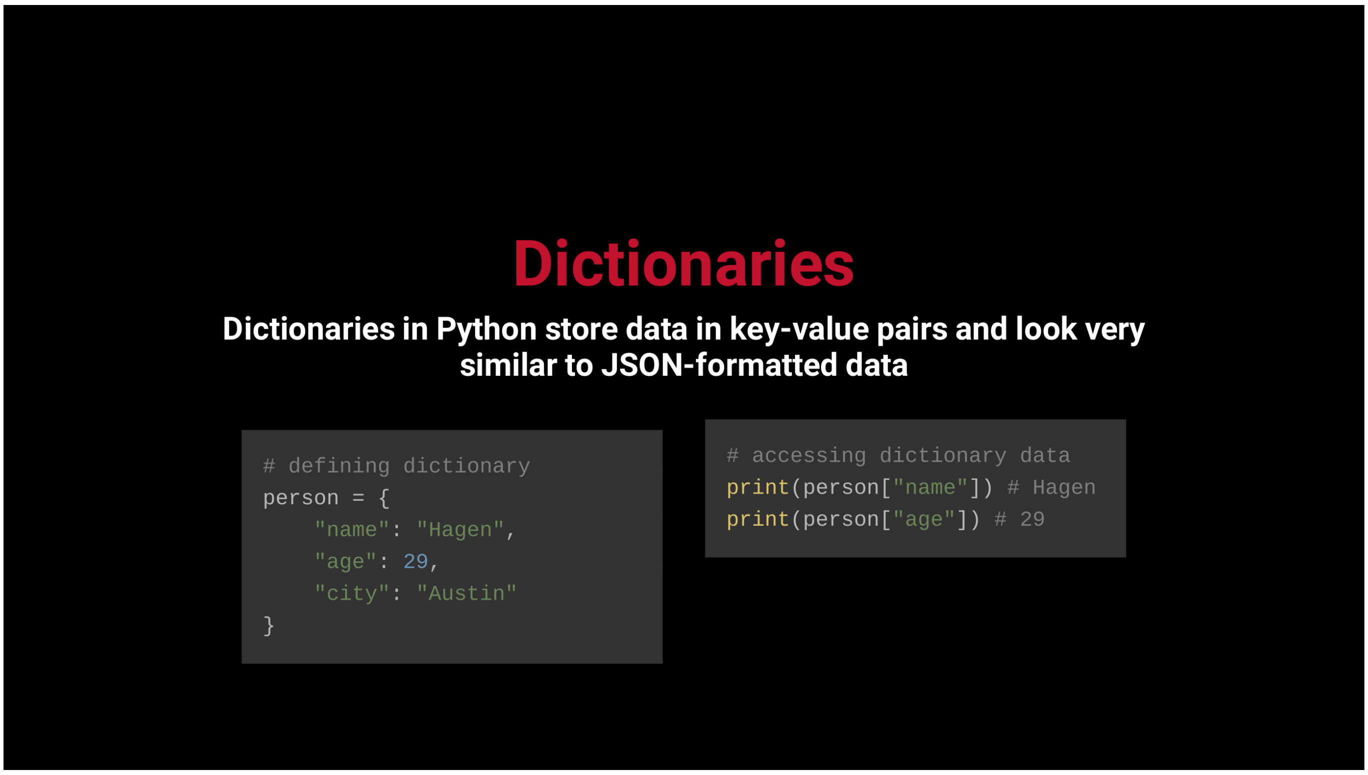
Lists are ordered collections of items, and they can hold any type of data.

```
fruits = ["apple", "banana", "cherry"]
```

Items in lists can be accessed by their index, with indices starting from 0 for the first item.

```
print(fruits[0])
```

Slide 12: Dictionaries



Dictionaries

Dictionaries in Python store data in key-value pairs and look very similar to JSON-formatted data

```python
# defining dictionary
person = {
    "name": "Hagen",
    "age": 29,
    "city": "Austin"
}
```

```python
# accessing dictionary data
print(person["name"]) # Hagen
print(person["age"]) # 29
```

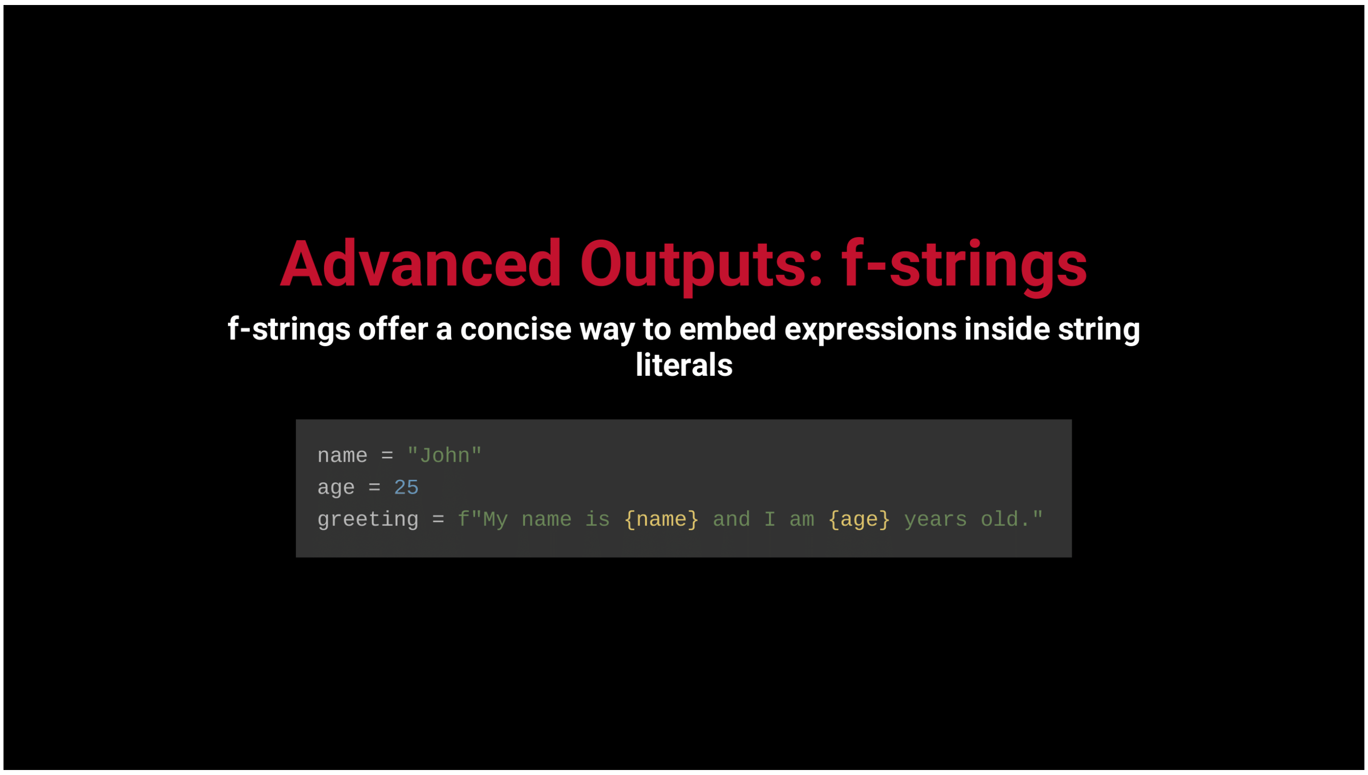
Dictionaries in Python store data in key-value pairs and look very similar to JSON-formatted data:

```python
person = {
    "name": "Hagen",
    "age": 29,
    "city": "Austin"
}
```

Values in a dictionary can be accessed using their respective keys:

```python
print(person["name"]) # Hagen
print(person["age"]) # 29
```

Slide 13: Advanced Outputs: F-Strings (Formatted String Literals)



Advanced Outputs: f-strings

f-strings offer a concise way to embed expressions inside string literals

```
name = "John"
age = 25
greeting = f"My name is {name} and I am {age} years old."
```

Introduced in Python 3.6, f-strings offer a concise way to embed expressions inside string literals. They are prefixed with an 'f' and use curly braces {} to embed Python expressions within the string.

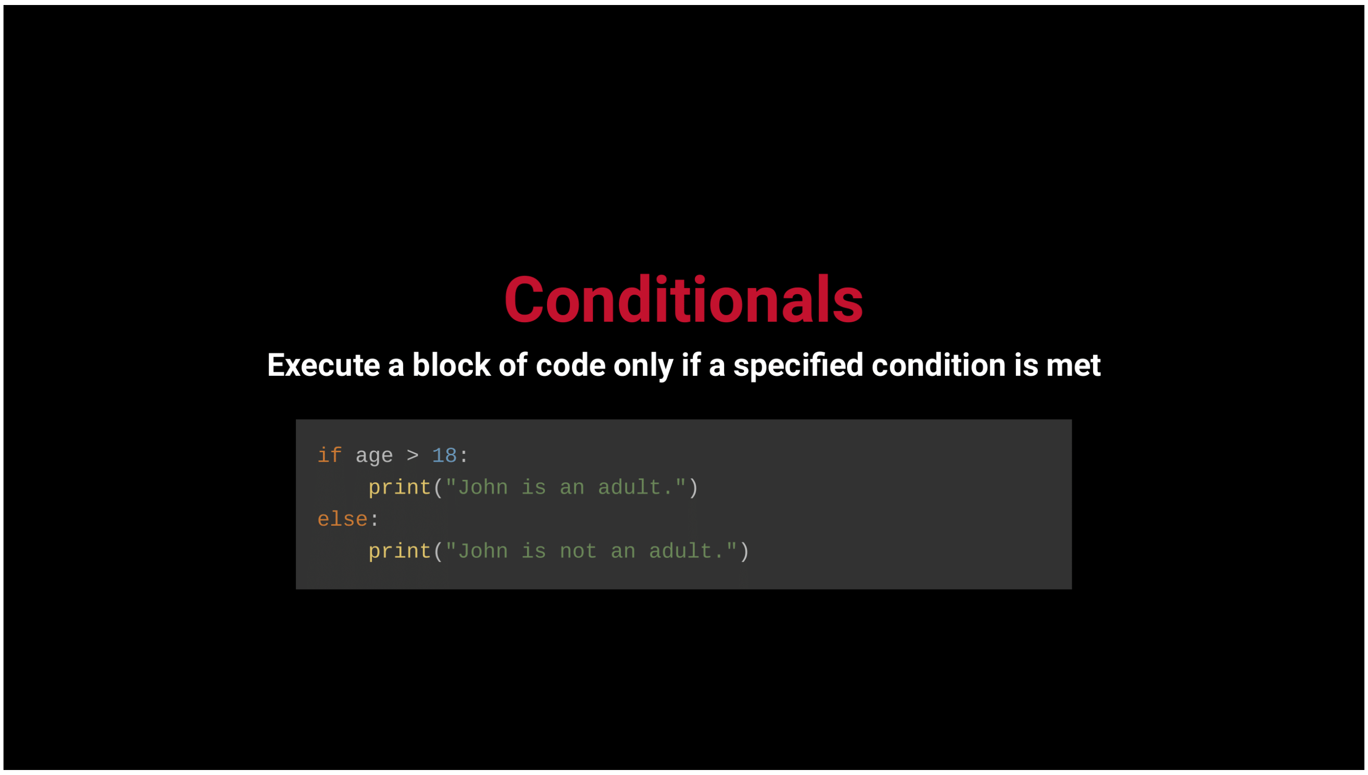
```
name = "John"
age = 25
greeting = f"My name is {name} and I am {age} years old."
print(greeting)  # Output: My name is John and I am 25 years old.
```

You can also perform operations within the curly braces of an f-string.

```
double_age = f"Twice my age is {age * 2}."
print(double_age)  # Output: Twice my age is 50.
```

F-strings provide a readable and convenient way to include variable values and expressions directly within strings, making code cleaner and more intuitive.

Slide 14: Conditionals



Conditionals

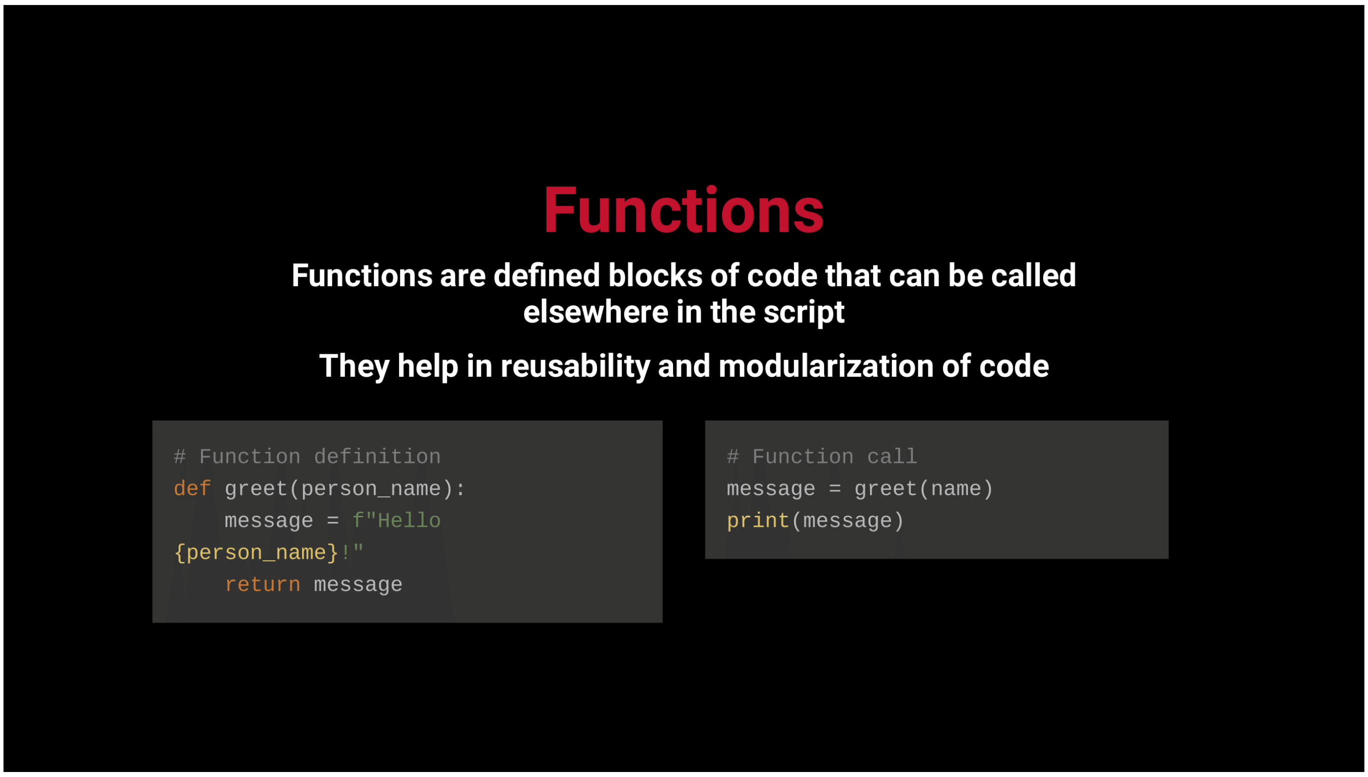
Execute a block of code only if a specified condition is met

```
if age > 18:
    print("John is an adult.")
else:
    print("John is not an adult.")
```

Conditionals allow for the execution of a block of code only if a specified condition is met.

```
if age > 18:
    print("John is an adult.")
else:
    print("John is not an adult.")
```

Slide 15: Functions



Functions

Functions are defined blocks of code that can be called elsewhere in the script

They help in reusability and modularization of code

```
# Function definition
def greet(person_name):
    message = f"Hello
{person_name}!"
    return message
```

```
# Function call
message = greet(name)
print(message)
```

- **Definition**: Functions are blocks of organized and reusable code designed to perform a specific task. They are a fundamental concept in programming, allowing for modularity and code reuse.

```
def function_name(parameters):
    """docstring: provides a brief explanation of what the function does, the
input(s), and the output(s)"""
    # function body
    return output
```

- **Parameters and Arguments**
  - Parameters are the names listed in the function's definition.
  - Arguments are the real values passed to the function when it's called.

```
def greet(person_name):
    message = f"Hello, {person_name}!"
    return message
```

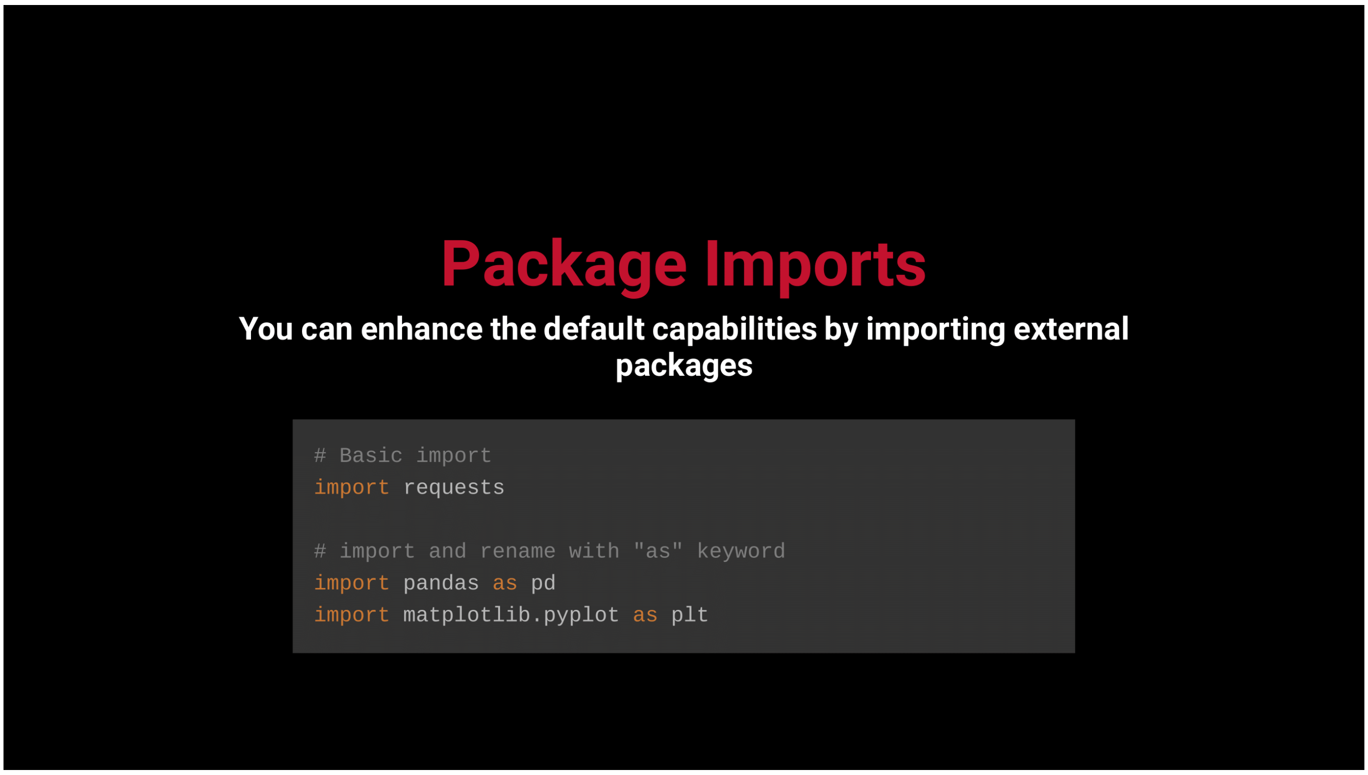
- **Return Statement**: The return keyword is used to exit a function and return a value.

```
def add(x, y):
    return x + y
```

- **Variable Scope**
  - Local Variables: Variables declared inside a function have a local scope, meaning they can only be accessed within that function
  - Global Variables: Variables declared outside of the function (or in global scope) can be accessed inside or outside of the function

```
x = 10  # This is a global variable

def check_value():
    y = 5  # This is a local variable
    return x + y  # Can access global variable 'x' inside this function
```

Slide 16: Package Imports



Package Imports

You can enhance the default capabilities by importing external packages

```python
# Basic import
import requests

# import and rename with "as" keyword
import pandas as pd
import matplotlib.pyplot as plt
```

In Python, we can enhance the default capabilities by importing external packages. This is done using the import keyword. In the script, two packages are imported:

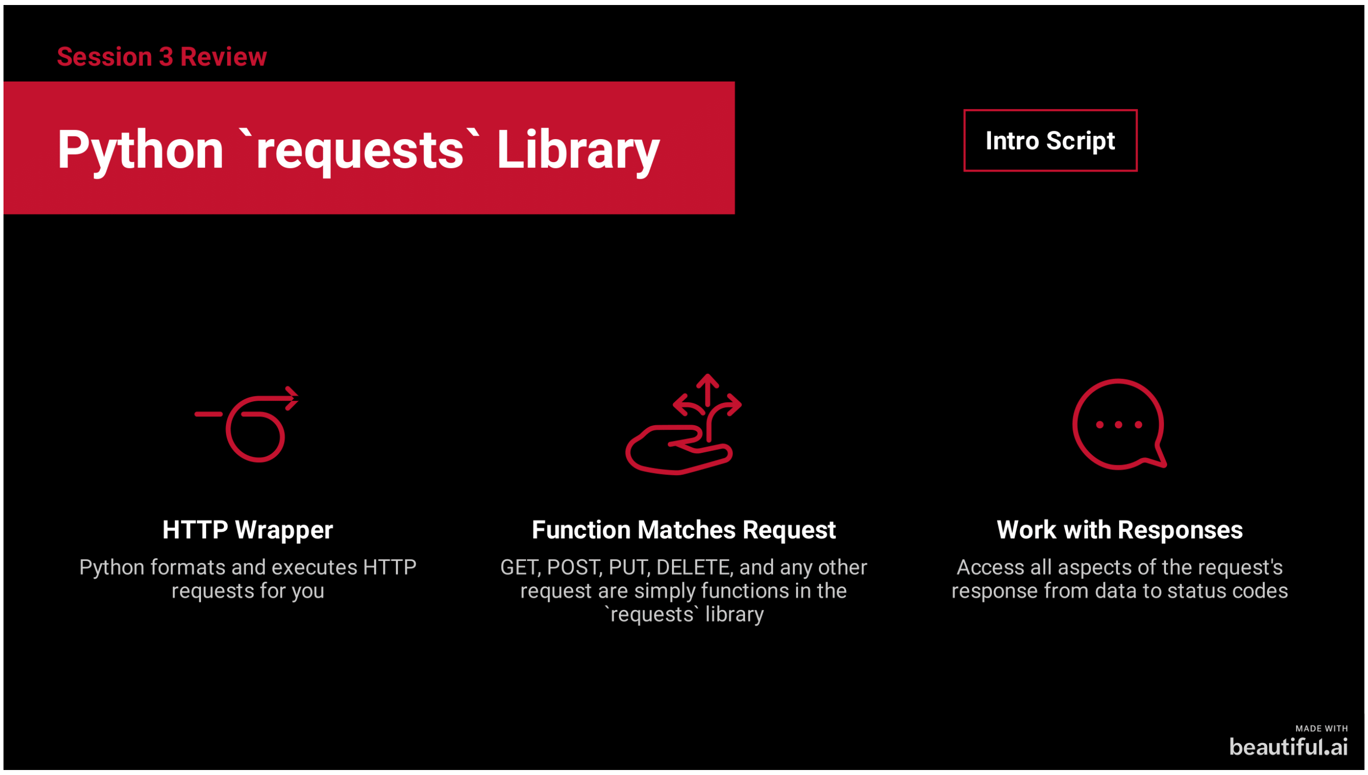
- requests: Used for making HTTP requests.

```python
import requests
```

- pandas (often imported as pd): Used for data manipulation and analysis.

```python
import pandas as pd
```

Slide 17: Python `requests` Library



Session 3 Review

Python `requests` Library

Intro Script

HTTP Wrapper
Python formats and executes HTTP requests for you

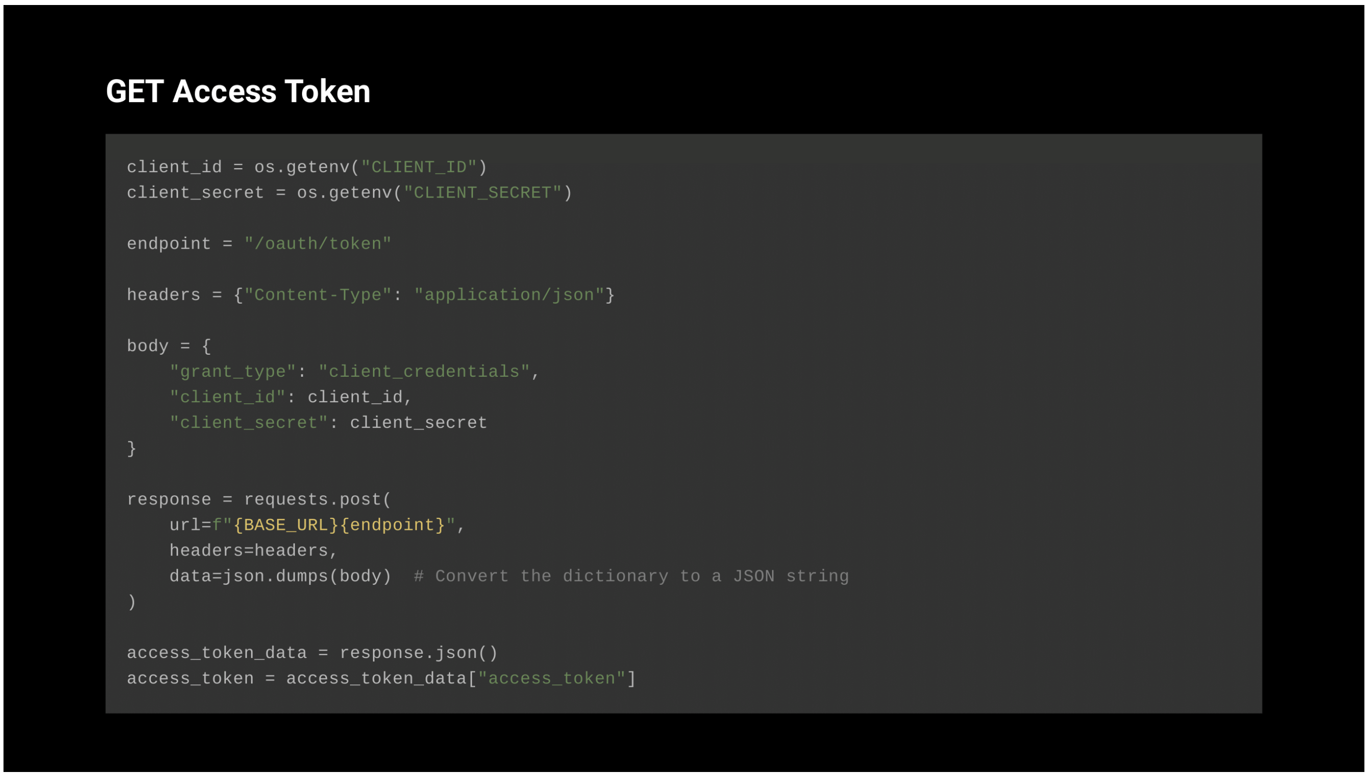
Function Matches Request
GET, POST, PUT, DELETE, and any other request are simply functions in the `requests` library

Work with Responses
Access all aspects of the request's response from data to status codes

MADE WITH beautiful.ai

1. **HTTP Wrapper**: The requests library is a popular Python package used for making HTTP requests. It abstracts the complexities of making requests behind a simple API, providing an intuitive way to send HTTP requests and handle responses.
2. **Methods**: It supports all major HTTP methods like GET, POST, PUT, DELETE, etc., through simple method calls.
3. **Response Handling**: Responses from servers can be easily parsed, and the library provides convenient methods to extract useful information, such as `response.text`, `response.json()`, and `response.status_code`.

Slide 18: POST Access Token in Procore



GET Access Token

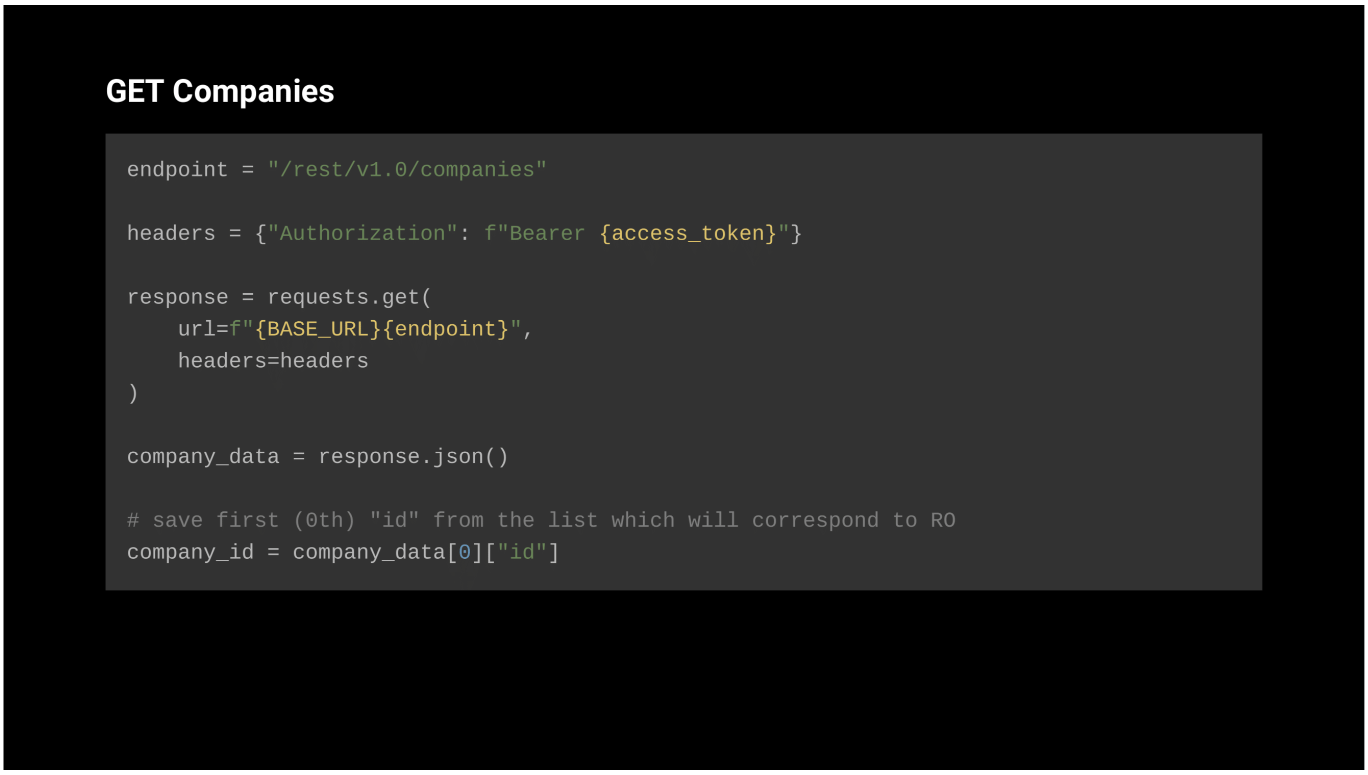
```python
client_id = os.getenv("CLIENT_ID")
client_secret = os.getenv("CLIENT_SECRET")

endpoint = "/oauth/token"

headers = {"Content-Type": "application/json"}

body = {
    "grant_type": "client_credentials",
    "client_id": client_id,
    "client_secret": client_secret
}

response = requests.post(
    url=f"{BASE_URL}{endpoint}",
    headers=headers,
    data=json.dumps(body)  # Convert the dictionary to a JSON string
)

access_token_data = response.json()
access_token = access_token_data["access_token"]
```

This snippet shows you how to create and access token for your Procore App

```
client_id = os.getenv("CLIENT_ID")
client_secret = os.getenv("CLIENT_SECRET")

endpoint = "/oauth/token"

headers = {"Content-Type": "application/json"}

body = {
    "grant_type": "client_credentials",
    "client_id": client_id,
    "client_secret": client_secret
}

response = requests.post(
    url=f"{BASE_URL}{endpoint}",
    headers=headers,
    data=json.dumps(body)  # Convert the dictionary to a JSON string
)

access_token_data = response.json()
access_token = access_token_data["access_token"]
```

1. `client_id = os.getenv("CLIENT_ID")`: Retrieves the value of the environment variable "CLIENT_ID" and assigns it to the client_id variable.
2. `client_secret = os.getenv("CLIENT_SECRET")`: Retrieves the value of the environment variable "CLIENT_SECRET" and assigns it to the client_secret variable.
3. `endpoint = "/oauth/token"`: Specifies the endpoint for obtaining an OAuth token.
4. `headers = {...}`: Defines a dictionary that indicates the content being sent is in JSON format.
5. `body = {...}`: Constructs the data payload, including the grant type and the client's ID and secret credentials.
6. `response = requests.post(...)`: Sends a POST request to the composed URL (BASE_URL + endpoint), with the defined headers and the body converted to a JSON string.
7. `access_token_data = response.json()`: Parses the response, which is expected to be in JSON format, into a Python dictionary. 8 `access_token = access_token_data["access_token"]`: Extracts the value associated with the key "access_token" from the parsed response and assigns it to the access_token variable.

Slide 19: GET All Companies



GET Companies

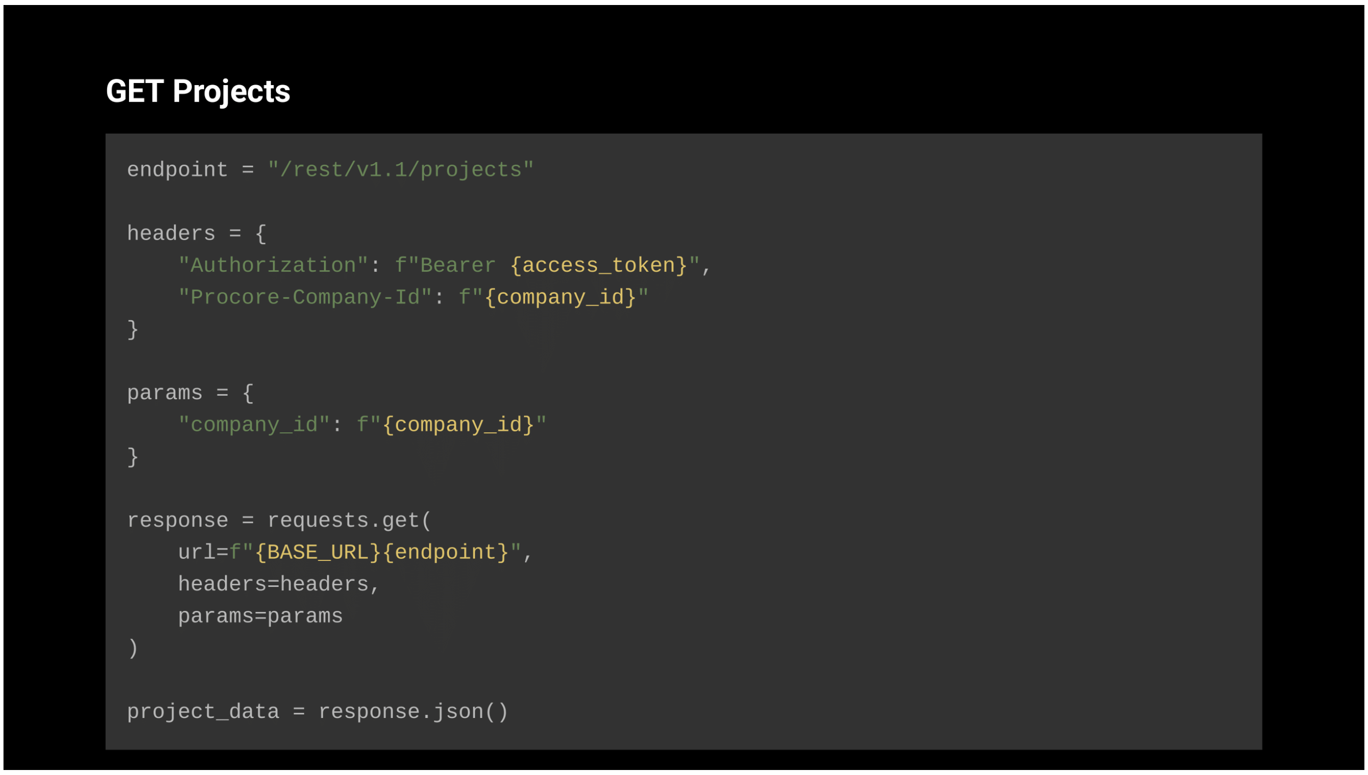
```python
endpoint = "/rest/v1.0/companies"

headers = {"Authorization": f"Bearer {access_token}"}

response = requests.get(
    url=f"{BASE_URL}{endpoint}",
    headers=headers
)

company_data = response.json()

# save first (0th) "id" from the list which will correspond to RO
company_id = company_data[0]["id"]
```

This snippet gets the companies that the app has been downloaded to.

```python
endpoint = "/rest/v1.0/companies"

headers = {"Authorization": f"Bearer {access_token}"}

response = requests.get(
    url=f"{BASE_URL}{endpoint}",
    headers=headers
)

company_data = response.json()

# save first (0th) "id" from the list which will correspond to RO
company_id = company_data[0]["id"]
```

1. `endpoint = "/rest/v1.0/companies"`: Specifies the endpoint for fetching company data.
2. `headers = {...}: Defines a dictionary containing an Authorization header, which uses the previously retrieved access_token.
3. `response = requests.get(...)`: Sends a GET request to the composed URL (BASE_URL + endpoint), with the defined headers.
4. `company_data = response.json()`: Parses the response, which is expected to be in JSON format, into a Python dictionary or list (based on the response structure).
5. `company_id = company_data[0]["id"]`: Extracts the "id" of the first (0th) company in the retrieved data list and assigns it to the company_id variable.

Slide 20: GET All Projects within a Company



GET Projects

```
endpoint = "/rest/v1.1/projects"

headers = {
    "Authorization": f"Bearer {access_token}",
    "Procore-Company-Id": f"{company_id}"
}

params = {
    "company_id": f"{company_id}"
}

response = requests.get(
    url=f"{BASE_URL}{endpoint}",
    headers=headers,
    params=params
)

project_data = response.json()
```

This snippet pulls all the projects that have granted access to your app.

```
endpoint = "/rest/v1.1/projects"

headers = {
    "Authorization": f"Bearer {access_token}",
    "Procore-Company-Id": f"{company_id}"
}

params = {
    "company_id": f"{company_id}"
}

response = requests.get(
    url=f"{BASE_URL}{endpoint}",
    headers=headers,
    params=params
)

project_data = response.json()
```

1. `endpoint = "/rest/v1.1/projects"`: Specifies the endpoint for fetching project data.
2. `headers = {...}`: Defines a dictionary for headers, including the previously retrieved access_token for authentication and specifying which company's projects to fetch.
3. `params = {"company_id": f"{company_id}"}`: Sets up query parameters to include the company_id in the request.
4. `response = requests.get(...)`: Sends a GET request to the composed URL (BASE_URL + endpoint), using the headers and parameters defined.

5. `project_data = response.json()`: Parses the response, expected to be in JSON format, into a Python dictionary or list (based on the response structure).

🏆 Challenge

▶ Click to Expand

Create a GET request in Python (Slide 21)

Using The Cat API, create a GET request to get a random cat image.

- [Documentation}(https://documenter.getpostman.com/view/5578104/RWgqUxxh#997f5b37-79cc-49a4-8c11-ddf24b72a4d9) on the `/images/search` endpoint
- The code you will complete is below:

```
def get_image():
    """
    Gets a random cat image

    Link to Documentation:
https://documenter.getpostman.com/view/5578104/RWgqUxxh#997f5b37-79cc-49a4-8c11-
ddf24b72a4d9
    """

    # Define the full URL
    get_url = ""

    # Define the header(s)
    # Use the following API Key:
live_NMNC1lcbODoViW3HxtYkiIstzdDd8wN2e8tlHLM6QyDGSKTA1NUGGdqEGP7UOoBm
    get_headers = {}

    # Create the request using the correct method from the requests library
    #response =

    # Print the URL to the image
    print("Challenge 1: Below is the URL to a random cat image")
    print(response.json()[0]["url"])
```

Some Help:

1. Define the `get_url` string variable using the full URL available in the documentation under the endpoint (should start with "https")
2. Define the `get_headers` dictionary variable. You need only one header related to the API Key which is provided in the code.
3. Create the GET request using the `requests.get()` function. For this request, you only need to provide values for the `url` and `headers` input parameters.

Create a POST request in Python (Slide 22)

Use The Cat API to create a POST request that likes an image.

- Documentation on the `/favourites` endpoint
- The image you will be liking - that is the Chicken Nugget
- The code you will complete is below:

```
def favorite_image():
    """
    Favorites a cat image

    Link to Documentation:
https://documenter.getpostman.com/view/5578104/RWgqUxxh#ae1b5e8f-ca63-4de8-a715-
f4944f4cec07
    """

    # Define the full URL
    post_url = ""

    # Define the header(s)
    # Use the following API Key:
live_NMNC1lcbODoViW3HxtYkiIstzdDd8wN2e8tlHLM6QyDGSKTA1NUGGdqEGP7UOoBm
    post_headers = {}

    # Define the data to send. You need to only use the `image_id` key and use the
ID: h-bMdWYmd
    post_data = {}

    # Create the request using the correct method from the requests library
    #response =

    #
    print("Challenge 2: You should get a 'SUCCESS' message below")
    print(response.text)
```

Some Help:

1. Define the `post_url` string variable using the full URL available in the documentation under the endpoint (should start with "https")
2. Define the `post_headers` dictionary variable. You need only one header related to the API Key which is provided in the code. This should match what you did before in the GET request.
3. Define the `post_data` dictionary variable which will include a key called `image_id` with a value of `h-bMdWYmd`.
4. Create the POST request using the `requests.post()` function. For this request, you need to provide three values: the `url`, the `headers`, and the `json` input parameters. For the `json` parameter, you will provide the data you created earlier.

Break (10 minutes)

👐 Hands-On: API Workshop

▶ Click to Expand

Slide 24: Software Use

Discussion Prompt: What are the main software applications or tools you use daily?

Tasks:

- Jot down at least 3 software tools each person uses often in their daily workflow
- Share your list with others to get inspired or help finish our their list

Goal: Create a consolidated list of software tools crucial to their tasks and workflows.

Slide 25: Find Pain Points

Discussion Prompt: What are some minor annoyances or inefficiencies you've noticed with these software tools?

Tasks:

- For each application, list 2 simple tasks you wish the tool could do
- For each application, list 1 major task/workflow you would like

Goal: Pinpoint areas where APIs could offer improvements or solutions.

Slide 26: Explore API Documentation

Discussion Prompt: What can APIs do for you?

Tasks:

- For each application, find if they have an API. If so, find their documentation.
- Skim through the documentation to see the different functionalities offered.
- Identify at least 1 functionality that could solve some of your pain points.
- Identify at least 1 functionality that could solve an issue you hadn't thought of!

Goal: Familiarize with the kind of operations and tasks the API supports.

Slide 27: Brainstorm API Solutions

Discussion Prompt: What process could you create to solve your issue?

Tasks

- Focus on one particular task you believe is important and that could be solved with the API.
- Brainstorm the process or workflow and be specific!
- If you have time, try another task or consult with others.

Goal: Create an outline of how the API could solve an issues you currently face.

Slide 28: Open Discussion

Discussion Prompt: How can others help refine your solution?

Tasks

- Discuss one solution you developed with another person
- Focus on what might be missing
- Are there any other considerations for the process?

Goal: Strengthen your process.
