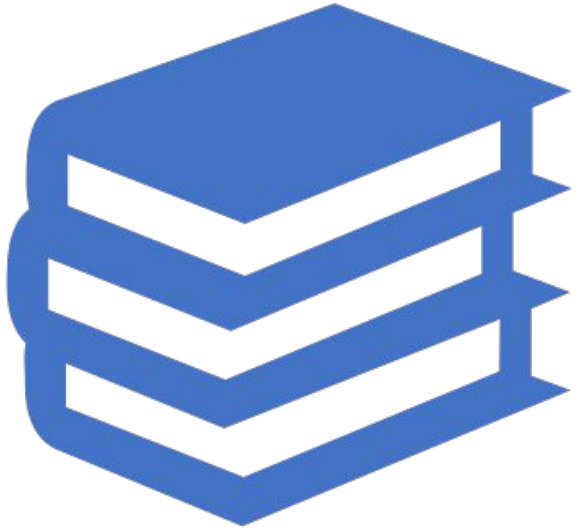
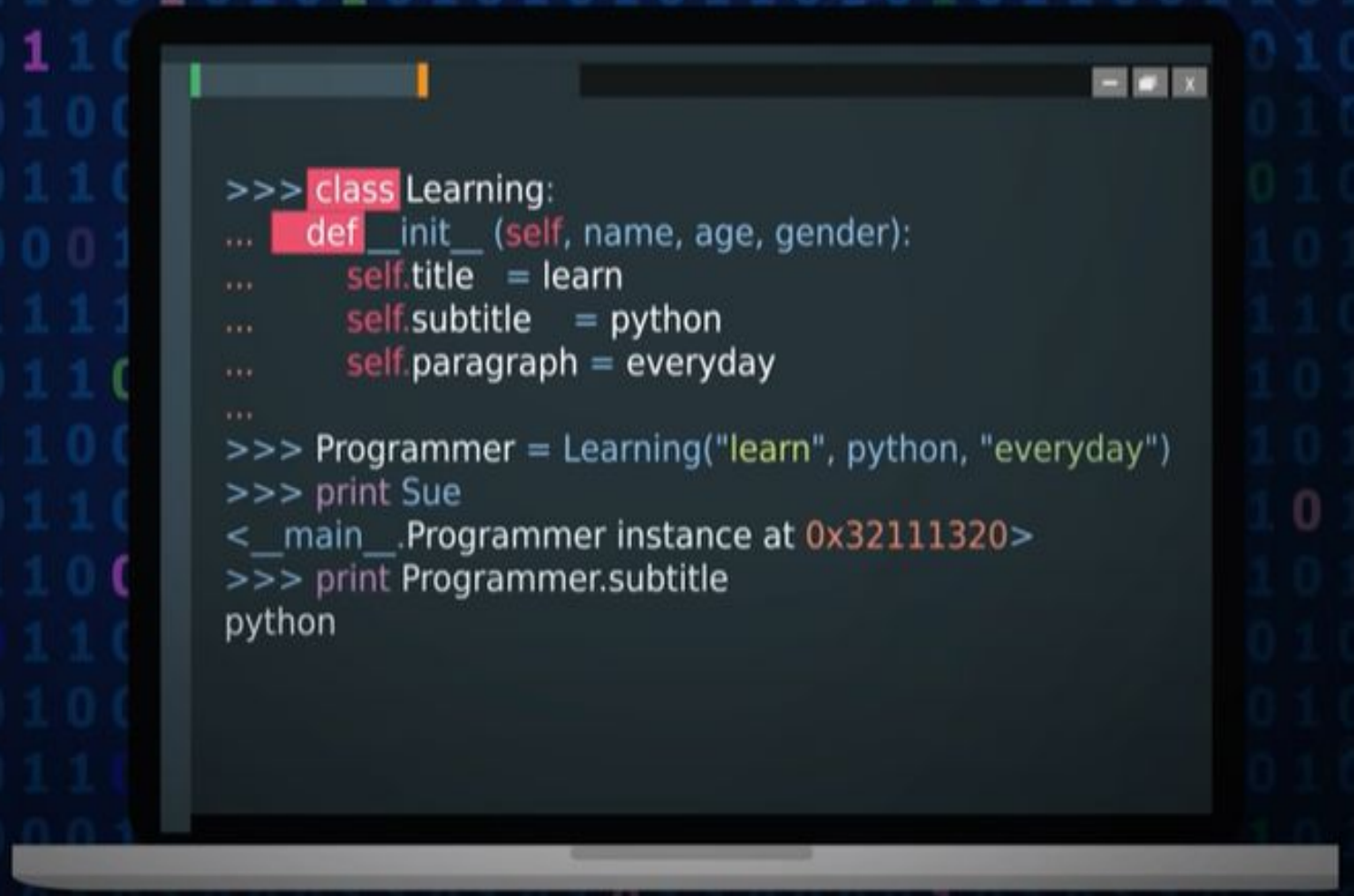


Python Lists, Tuples, & Dictionaries

Lists, Tuples and Dictionaries in Python

Lists in Python

Python Lists



What's in it for you?

- What are lists in Python?
- Creating lists
- Accessing elements in lists
- Operations on lists
- Methods with lists
- Built-in functions with lists
- Exercise

What are lists ?

A list can be defined as a collection of objects, values or items of different types and these collections are enclosed within the square brackets[] and separated by commas (,)

List1 = [1, 'Adam' , 107, 'USA']

Characteristics of List

The lists are ordered

Lists can store various types of elements

Elements of the list can be accessed by index

Lists are mutable

Lists allow duplicate elements

[39]:
```
#  Basic List:
num = [1, 2, 3, 4, 5]
print(num)
```

```
[1, 2, 3, 4, 5]
```

[40]:
```
# List with Mixed Data Types:
mixed_list = [1, "hello", 3.14, True]
print(mixed_list)
```

```
[1, 'hello', 3.14, True]
```

[41]:
```
# Nested Lists:
nested_list = [[1, 2, 3], ["a", "b", "c"], [True, False]]
print(nested_list)
```

```
[[1, 2, 3], ['a', 'b', 'c'], [True, False]]
```

[42]:
```
# Creating an Empty List:
empty_list = []
print(empty_list)
```

```
[]
```

[43]:
```
#  Using the list() Constructor:
constructed_list = list("Python")
print(constructed_list)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

[44]:
```
# Repetition of Elements:
repeated_list = [0] * 3
print(repeated_list)
```

```
[0, 0, 0]
```

[45]:
```python
# Creating a Range of Numbers:
number_range = list(range(1, 6))
print(number_range)
```

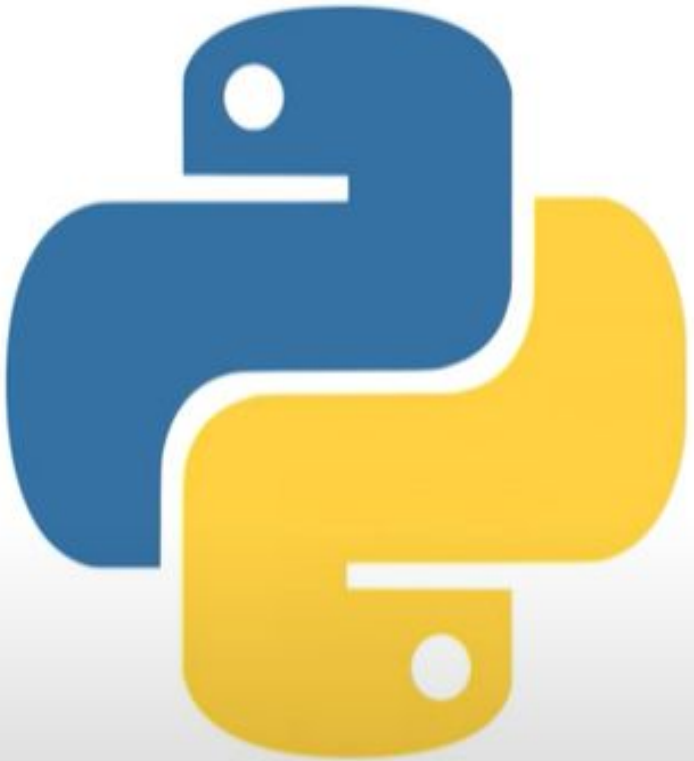
```
[1, 2, 3, 4, 5]
```

[46]:
```python
# List Comprehension:
squares = [x**2 for x in range(5)]
print(squares)
```

```
[0, 1, 4, 9, 16]
```

What's in it for you?

- What are lists in Python?
- Creating lists
- Accessing elements in lists
- Operations on lists
- Methods with lists
- Built-in functions with lists
- Exercise



[47]:
```
# Accessing a Single Element by Index:
my_list = [10, 20, 30, 40, 50]
first_element = my_list[0]
third_element = my_list[2]
print(first_element, third_element)
```

```
10 30
```

[48]:
```
#  Negative Indexing (Accessing Elements from the End):
my_list = [10, 20, 30, 40, 50]
last_element = my_list[-1]
second_to_last = my_list[-2]
print(last_element, second_to_last)
```

```
50 40
```

[49]:
```
# Slicing to Access a Sublist:
my_list = [10, 20, 30, 40, 50]
sublist = my_list[1:4]
print(sublist)
```

```
[20, 30, 40]
```

[50]:
```python
# Modifying Elements through Indexing:
my_list = [10, 20, 30, 40, 50]
my_list[2] = 35
print(my_list)
```

```
[10, 20, 35, 40, 50]
```

[51]:
```python
# Accessing Nested Lists:
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
element = nested_list[1][2]
print(element)
```

```
6
```

[52]:
```python
# Checking if an Element Exists in a List:
my_list = [10, 20, 30, 40, 50]
element_to_check = 30
if element_to_check in my_list:
    print(f"{element_to_check} exists in the list.")
```

```
30 exists in the list.
```

What's in it for you?

- What are lists in Python?
- Creating lists
- Accessing elements in lists
- Operations on lists
- Methods with lists
- Built-in functions with lists
- Exercise

Append	List1.append(elem)
Insert	List1.insert(index,elem)
Extend	List1.extend(list2)
Index	List1.index(elem)
Remove	List1.remove(elem)
Sort	List1.sort()
Reverse	List1.reverse()

[53]:
```
# Append:
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
```

```
[1, 2, 3, 4]
```

[54]:
```
#  Extend:
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list1.extend(list2)
print(list1)
```

```
[1, 2, 3, 4, 5, 6]
```

[55]:
```
#  Insert:
my_list = [1, 2, 3]
my_list.insert(1, 5)
print(my_list)
```

```
[1, 5, 2, 3]
```

[56]:
```
# Remove:
my_list = [1, 2, 3, 4, 2]
my_list.remove(2)
print(my_list)
```

```
[1, 3, 4, 2]
```

[57]:
```
# Pop:
my_list = [1, 2, 3]
popped_element = my_list.pop(1)
print(my_list, popped_element)
```

```
[1, 3] 2
```

[58]:
```
# Clear:
my_list = [1, 2, 3]
my_list.clear()
print(my_list)
```

```
[]
```

[59]:
```
#  Index:
my_list = [10, 20, 30, 40, 50]
index_of_30 = my_list.index(30)
print(index_of_30)
```

```
2
```

[60]:
```
#  Count:
my_list = [1, 2, 3, 2, 4, 2]
count_of_2 = my_list.count(2)
print(count_of_2)
```

```
3
```

[61]:
```
# Sort:
my_list = [4, 1, 3, 2]
my_list.sort()
print(my_list)
```

```
[1, 2, 3, 4]
```

[62]:
```
#  Reverse:
my_list = [1, 2, 3, 4]
my_list.reverse()
print(my_list)
```

```
[4, 3, 2, 1]
```

[63]:
```
#  Copy:
original_list = [1, 2, 3]
copied_list = original_list.copy()
print(copied_list)
```

```
[1, 2, 3]
```

What's in it for you?

- What are lists in Python?
- Creating lists
- Accessing elements in lists
- Operations on lists
- Methods with lists
- Built-in functions with lists
- Exercise

[1]:
```
# len():
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length)
```

```
5
```

[3]:
```
# sum():
my_list = [1, 2, 3, 4, 5]
total = sum(my_list)
print(total)
```

```
15
```

[4]:
```
# min() and max():
my_list = [1, 2, 3, 4, 5]
minimum = min(my_list)
maximum = max(my_list)
print(minimum, maximum)
```

```
1 5
```

[5]:
```python
# sorted():
my_list = [4, 1, 3, 2]
sorted_list = sorted(my_list)
print(sorted_list)
```

```
[1, 2, 3, 4]
```

[6]:
```python
# any() and all():
bool_list = [True, False, True, True]
any_true = any(bool_list)
all_true = all(bool_list)
print(any_true, all_true)
```

```
True False
```

[7]:
```python
# enumerate():
my_list = ['a', 'b', 'c']
for index, value in enumerate(my_list):
    print(index, value)
```

```
0 a
1 b
2 c
```

[8]:
```python
# zip():
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
zipped_list = list(zip(list1, list2))
print(zipped_list)
```

```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

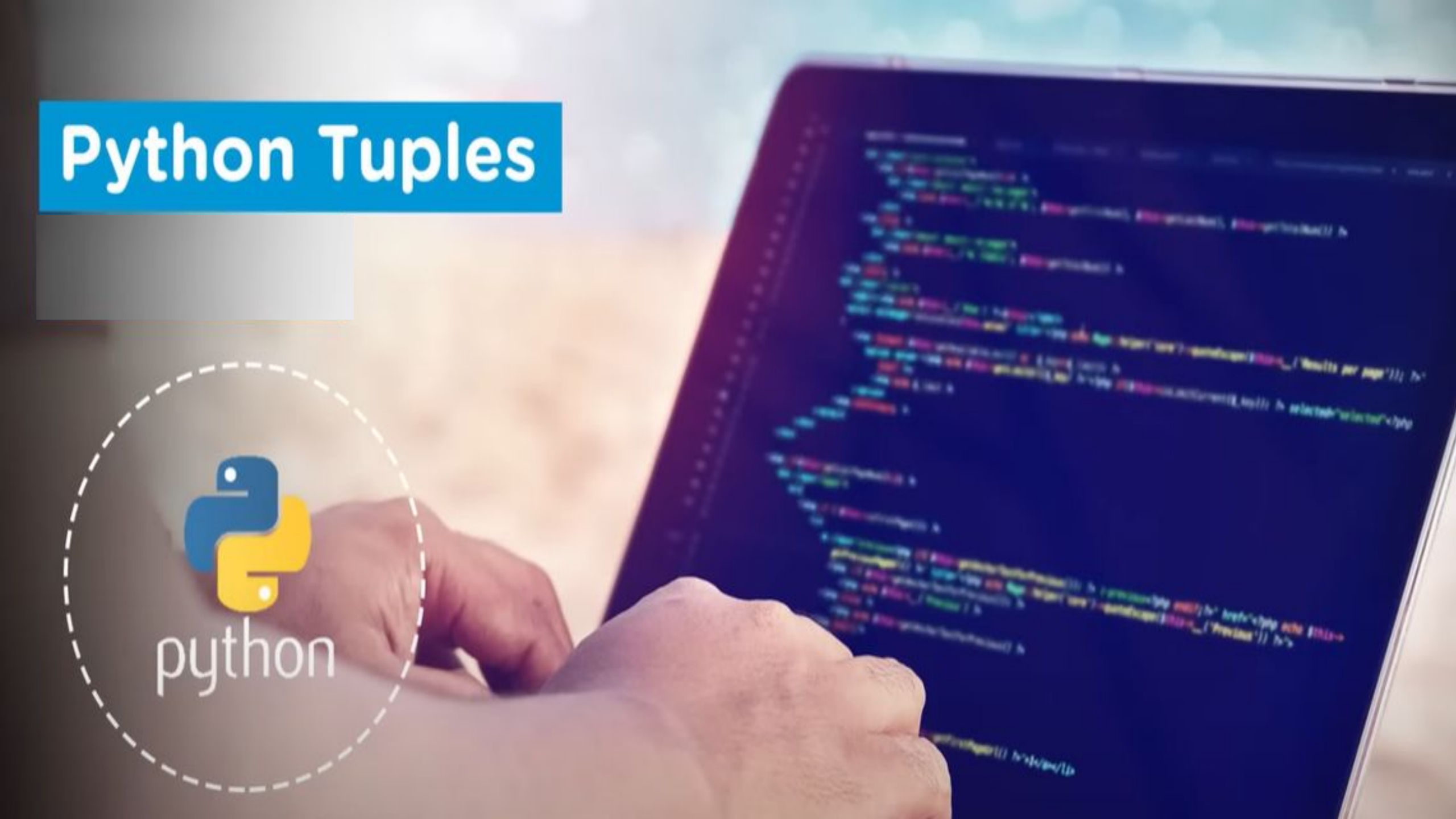
[9]:
```python
# map():
my_list = [1, 2, 3]
squared_list = list(map(lambda x: x**2, my_list))
print(squared_list)
```

```
[1, 4, 9]
```

[10]:
```python
# filter():
my_list = [1, 2, 3, 4, 5]
filtered_list = list(filter(lambda x: x % 2 == 0, my_list))
print(filtered_list)
```

```
[2, 4]
```

Python Tuples



What are tuples?

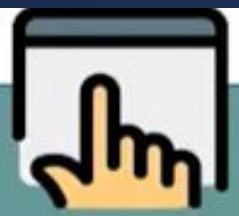
A tuple can be defined as a collection of objects, values or items of different types and these collections are enclosed within the circle brackets() and separated by commas (,)

Tup1 = (1, 'Adam' , 107, 'USA')

Characteristics of Tuples

The tuples are ordered

Tuples can store various types of elements



Elements of the tuples can be accessed by index

Tuples are immutable

Tuples allow duplicate elements

Creating tuples

[11]:
```python
# Basic Tuple:
my_tuple = (1, 2, 3)
print(my_tuple)
```

```
(1, 2, 3)
```

[12]:
```python
# Tuple with Mixed Data Types:
mixed_tuple = (1, "hello", 3.14, True)
print(mixed_tuple)
```

```
(1, 'hello', 3.14, True)
```

[13]:
```python
#  Nested Tuples:
nested_tuple = ((1, 2, 3), ("a", "b", "c"), (True, False))
print(nested_tuple)
```

```
((1, 2, 3), ('a', 'b', 'c'), (True, False))
```

Creating tuples

[16]:
```
# Single-Element Tuple:
single_element_tuple = (42,)
print(single_element_tuple)
```

```
(42,)
```

[17]:
```
# Using the tuple() Constructor:
constructed_tuple = tuple([1, 2, 3])
print(constructed_tuple)
```

```
(1, 2, 3)
```

[18]:
```
# Creating an Empty Tuple:
empty_tuple = ()
print(empty_tuple)
```

```
()
```

Creating tuples

[20]:
```
#  Unpacking Tuples:
""" In this example, the values from the tuple coordinates
are unpacked into the variables x and y. """
coordinates = (4, 5)
x, y = coordinates
print(x, y)
```

```
4 5
```

[21]:
```
# Creating a Tuple with range():
number_tuple = tuple(range(1, 6))
print(number_tuple)
```

```
(1, 2, 3, 4, 5)
```

Difference between tuple and list

Feature	List	Tuple
Mutability	Mutable	Immutable
Syntax	Defined using square brackets `[]`	Defined using parentheses `()`
Modification Methods	`append()`, `extend()`, `insert()`, `remove()`, `pop()`, etc.	Limited methods due to immutability
Use Cases	Dynamic collections, sequences that need to be modified	Fixed collections, constant data
Performance	Slightly larger memory overhead, may be less efficient for iteration	Slightly more memory-efficient, better for iteration
Example	`my_list = [1, 2, 3]`	`my_tuple = (1, 2, 3)`

Differences between tuples and lists

Feature	List	Tuple
Mutability	Mutable	Immutable

In [12]:
```
1 list1=[1,2,3,4]
2 tuple1=(1,2,3,4)
3 list1.append(5)
4 print(list1)
```

```
[1, 2, 3, 4, 5]
```

In [13]:
```
1 tuple1.append(5)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-c9bc946a1e1f> in <module>()
----> 1 tuple1.append(5)

AttributeError: 'tuple' object has no attribute 'append'
```

Methods in Tuple

Index	Tup1.index(elem)
Slicing	Tup1[range]
Concatenation	Tup1 + Tup2
Repetition	Tup1*x
Count	Tup1.count(elem)

[22]:
```
#  Indexing::
my_tuple = (10, 20, 30, 40, 50)
element = my_tuple[2]  # Accessing the element at index 2
print(element)
```

```
30
```

[23]:
```
# Slicing::
my_tuple = (10, 20, 30, 40, 50)
subset = my_tuple[1:4]  # Slicing from index 1 to 3 (excluding 4)
print(subset)
```

```
(20, 30, 40)
```

[24]:
```
#  Concatenation:
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = tuple1 + tuple2
print(concatenated_tuple)
```

```
(1, 2, 3, 4, 5, 6)
```

[25]:
```
#  Repetition:
my_tuple = (1, 2)
repeated_tuple = my_tuple * 3  # Repeating the tuple three times
print(repeated_tuple)
```

```
(1, 2, 1, 2, 1, 2)
```

[26]:
```
# Count:
my_tuple = (1, 2, 2, 3, 2, 4)
count_of_2 = my_tuple.count(2)  # Counting occurrences of 2 in the tuple
print(count_of_2)
```

```
3
```

Python Dictionaries

Dictionaries in Python

Python dictionary can be defined as a collection of objects, values or items of different types stored in key-value pair format. These multiple key-value pairs created are enclosed within the curly braces{}, and each key is separated from its value by the colon (:)

Dictionary is an unordered collection of data stored as a pair of key and value

Value

```
Dict1 = {'a' : 1, 'b' : 2, 'c' : 3}
```

Key

Syntax:

```
variable_name={key1 : value1, key2 : value2,..}
```

Characteristics of Dictionaries

The Dictionaries are ordered from Python version 3.7

Dictionaries can store various types of elements

Elements of the dictionaries can not be accessed by index

Dictionaries are mutable

Dictionaries doesn't allow duplicate elements

[27]:
```python
# Basic Dictionary:
# Creating a dictionary with key-value pairs
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}
print(my_dict)
```

```
{'name': 'John', 'age': 25, 'city': 'New York'}
```

[28]:
```python
# Using the dict() Constructor:
# Creating a dictionary using the dict() constructor
my_dict = dict(name='Alice', age=30, city='Paris')
print(my_dict)
```

```
{'name': 'Alice', 'age': 30, 'city': 'Paris'}
```

[29]:
```python
# Nested Dictionary:
# Creating a nested dictionary
employee = {
    'name': 'Bob',
    'age': 28,
    'department': {
        'name': 'IT',
        'location': 'Building A'
    }
}
print(employee)
```

```
{'name': 'Bob', 'age': 28, 'department': {'name': 'IT', 'location': 'Building A'}}
```

Methods in Dictionaries

clear	Dict1.clear()
get	Dict1.get(keyname)
keys	Dict1.keys()
pop	Dict1.pop(keyname)
popitem	Dict1.popitem()

Methods in Dictionaries

```
# clear():
# Creating a dictionary
my_dict = {'name': 'Alice', 'age': 25, 'city': 'Paris'}

# Clearing all items from the dictionary
my_dict.clear()
print(my_dict)
```

```
{}
```

```
# get():
# Creating a dictionary
my_dict = {'name': 'Bob', 'age': 30, 'city': 'New York'}

# Getting the value associated with the 'age' key
age_value = my_dict.get('age')
print(age_value)

# Attempting to get the value for a non-existent key with a default value
nonexistent_value = my_dict.get('salary', 0)
print(nonexistent_value)
```

```
30
0
```

```
# keys():
# Creating a dictionary
my_dict = {'name': 'Charlie', 'age': 22, 'city': 'London'}

# Getting a list of keys in the dictionary
keys_list = my_dict.keys()
print(keys_list)
```

```
dict_keys(['name', 'age', 'city'])
```

Methods in Dictionaries

```
# pop():
# Creating a dictionary
my_dict = {'name': 'David', 'age': 35, 'city': 'Berlin'}

# Removing and returning the value associated with the 'age' key
removed_age = my_dict.pop('age')
print(removed_age)
# Output: 35

# The dictionary after removing the 'age' key
print(my_dict)
# Output: {'name': 'David', 'city': 'Berlin'}
```

```
# popitem():
# Creating a dictionary
my_dict = {'name': 'Eva', 'age': 28, 'city': 'Madrid'}

# Removing and returning the last key-value pair from the dictionary
removed_item = my_dict.popitem()
print(removed_item)
# Output: ('city', 'Madrid')

# The dictionary after removing the last key-value pair
print(my_dict)
# Output: {'name': 'Eva', 'age': 28}
```

Methods in Dictionaries

```
# Adding a Single Element:
# Creating an empty dictionary
my_dict = {}

# Adding a key-value pair to the dictionary
my_dict['name'] = 'Alice'
print(my_dict)
# Output: {'name': 'Alice'}
```

```
# Adding Multiple Elements:
# Creating a dictionary with existing key-value pairs
my_dict = {'name': 'Bob', 'age': 30}

# Adding multiple key-value pairs using the update() method
my_dict.update({'city': 'New York', 'gender': 'Female'})
print(my_dict)
# Output: {'name': 'Bob', 'age': 30, 'city': 'New York', 'gender': 'Female'}
```

```
# Using the setdefault() Method:
# Creating a dictionary with existing key-value pairs
my_dict = {'name': 'Charlie', 'age': 22}

# Using setdefault() to add a key-value pair if the key does not exist
my_dict.setdefault('city', 'London')
print(my_dict)
# Output: {'name': 'Charlie', 'age': 22, 'city': 'London'}
```

Dictionary Comprehension

1. Create dictionaries from:
    - lists
    - tuples
    - dataframes

2. Modify dictionaries in place

3. Create complex data structures:
    - dictionary of lists
    - list of dictionaries

FOR LOOP APPROACH - ZIP

```
names = ['Mariya', 'Gendalf', 'Batman']
profs = ['programmer', 'wizard', 'superhero']

my_dict = {}
```

FOR LOOP APPROACH - ZIP

```
for (key, value) in zip(names, profs):
    my_dict[key] = value

print(my_dict)
```

```
>> {'Mariya': 'programmer', 'Gendalf': 'wizard', 'Batman': 'superhero'}
```

FOR LOOP APPROACH - RANGE

```
names = ['Mariya', 'Gendalf', 'Batman']
profs = ['programmer', 'wizard', 'superhero']

my_dict = {}
```

FOR LOOP APPROACH - RANGE

```
for i in range(3):
    my_dict[names[i]] = profs[i]

print(my_dict)
```

```
>> {'Mariya': 'programmer', 'Gendalf': 'wizard', 'Batman': 'superhero'}
```

Dictionary Comprehension

```
list1 = ['name', 'age', 'city']
list2 = ['Alice', 25, 'Paris']
my_dict ={}
for (key,value) in zip(list1, list2):
    my_dict[key] = value
print(my_dict)
```

```
{'name': 'Alice', 'age': 25, 'city': 'Paris'}
```

```
list1 = ['name', 'age', 'city']
list2 = ['Alice', 25, 'Paris']
my_dict ={}
for i in range(3):
    my_dict[list1[i]] = list2[i]
print(my_dict)
```

```
{'name': 'Alice', 'age': 25, 'city': 'Paris'}
```

```
list1 = ['name', 'age', 'city']
list2 = ['Alice', 25, 'Paris']
my_dict = {
    list1[i]:list1[i] for i in range(3)
}
print(my_dict)
```

```
{'name': 'name', 'age': 'age', 'city': 'city'}
```

Dictionary Comprehension

```python
# Creating a Dictionary from Tuples:
# Creating a list of tuples
data_tuples = [('name', 'Bob'), ('age', 30), ('city', 'New York')]

# Using dictionary comprehension to create a dictionary
my_dict = {k: v for k, v in data_tuples}
print(my_dict)
```

```
{'name': 'Bob', 'age': 30, 'city': 'New York'}
```

```python
# Creating a Dictionary from DataFrames (using pandas):
import pandas as pd

# Creating a DataFrame
data = {'name': ['Alice', 'Bob', 'Charlie'],
        'age': [25, 30, 22],
        'city': ['Paris', 'New York', 'London']}
df = pd.DataFrame(data)

# Using dictionary comprehension to create a dictionary from DataFrame columns
my_dict = {column: df[column].tolist() for column in df.columns}
print(my_dict)
```

```
{'name': ['Alice', 'Bob', 'Charlie'], 'age': [25, 30, 22], 'city': ['Paris', 'New York', 'London']}
```