Lecture slides - Week 15

Python - Advanced Techniques and Functional Programming

Dr. Aamir Akbar

Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

Contents

1. Python Advanced Builtin Functions

2. Generators

3. Decorators

Python Advanced Builtin Functions

Lambda Function

Lambda is an anonymous, small, and inline function defined using the `lambda` keyword. It's often used for simple operations where creating a named function is unnecessary. Lambda functions can take any number of arguments but can only have one expression.

```python
# standard function to add two numbers
def add_function(x, y):
    return x + y

print(add_function(3, 5))  # Output: 8
```

```python
# Equivalent Lambda function to add two numbers
add_lambda = lambda x, y: x + y

print(add_lambda(3, 5))  # Output: 8
```

The `add_lambda` takes two arguments `x` and `y` and returns their sum that is `x + y`.

map() Function

The map() function in Python is used to apply a given function to each item of an iterable (e.g., a list) and returns a map object, which is an iterator. It takes in a function and one or more iterables and applies the function to each element in the iterables.

```python
# Lambda function to convert Celsius to Fahrenheit
celsius_to_fahrenheit = lambda celsius: (celsius * 9/5) + 32

# List of Celsius temperatures
celsius_temps = [0, 10, 20, 30, 40]

# Using map to convert Celsius temperatures to Fahrenheit
fahrenheit_temps = map(celsius_to_fahrenheit, celsius_temps)

# Displaying the Fahrenheit temperatures using map
print(list(fahrenheit_temps))  # Output: [32.0, 50.0, 68.0, 86.0, 104.0]
```

The above example shows how map() applies the celsius_to_fahrenheit lambda function to each element in the celsius_temps list, returning a map object which, when converted to a list, gives the Fahrenheit temperatures corresponding to the Celsius values.

reduce() Function

`reduce()` function from the `functools` module in Python-3 is used for performing computation on a collection and returning the result.

```python
from functools import reduce

# Function to multiply two numbers
def multiply(x, y):
    return x * y

# List of numbers
numbers = [1, 2, 3, 4, 5]

# Using reduce to find the product of numbers
product = reduce(multiply, numbers)

print(product)  # Output: 120 (1 * 2 * 3 * 4 * 5)
```

The function passed to reduce() is applied to the first two elements of the list, then to the result and the next element in the list, and so on, until the list is finished.

zip() Function

The `zip()` function in Python is used to combine elements from multiple iterables (like lists) into tuples. It takes iterables as input and returns an iterator of tuples where the i-th tuple contains the i-th elements from each of the input iterables.

```python
# Two lists
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]

# Using zip to combine elements from lists into tuples
combined = zip(names, ages)

# Converting the zip object into a list of tuples
combined_list = list(combined)

print(combined_list)
# Output: [('Alice', 25), ('Bob', 30), ('Charlie', 35)]
```

Generators

What is a generator?

Generator in Python is a function that enable you to generate a sequence of values over time, rather than computing and storing all values at once like a regular function.

They are defined like regular functions but use the yield statement to return data one at a time, suspending their state between calls. This helps in efficiently dealing with large datasets or infinite sequences.

```python
def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# Using the generator to print Fibonacci numbers up to a certain limit
limit = 10
fib_gen = fibonacci_generator()

for _ in range(limit):
    print(next(fib_gen))
```

Decorators

What are decorators?

In Python, decorators are functions that modify the behavior of other functions or methods without changing their actual code. They allow you to add functionality to existing functions dynamically. Decorators are denoted by the @decorator_name syntax and are placed above the function they are decorating.

```python
# Decorator function
def decorator_function(func):
    def wrapper():
        print("Executing some code before the function")
        func()
        print("Executing some code after the function")
    return wrapper

# Function decorated with the decorator function
@decorator_function
def greet():
    print("Hello, world!")

# Calling the decorated function
greet()
```