Introduction To Python

Dr.Hajialiasgari

Tehran University
Of
Medical Science

November 17, 2024

TEHRAN UNIVERSITY
OF
MEDICAL SCIENCES

Lambda Functions

- A lambda function is a small, anonymous function that can have any number of arguments but only one expression. It's often used for short, simple operations where defining a full function might be unnecessary.
- `Characteristics:`
- No def keyword or name.
- Limited to one expression (e.g., no loops or multiple statements).
- Typically used in combination with functions like filter, map, and sorted.

lambda example

```
# Regular function
def multiply(x, y):
    return x * y

# Equivalent lambda function
multiply = lambda x, y: x * y

# Usage
result = multiply(4, 5)
# Output: 20
```

Filter

- The `filter()` function applies a filtering condition to an iterable (e.g., list, tuple) and returns only the elements that satisfy the condition.
- Returns: A filter object, which can be converted to a list or other iterable types.
- Use Cases: Extracting specific elements based on a criterion (e.g., filtering even numbers, removing None values).

filter Example

```
# Remove strings with less than 5 characters
words = ["apple", "kiwi", "banana", "pear"]
filtered_words = filter(lambda word: len(word) >= 5, words)

print(list(filtered_words))
# Output: ['apple', 'banana']
^^I
```

Map

- The `map()` function applies a transformation function to all elements in an iterable. Its often used when you need to perform the same operation on each element of a sequence.
- Returns: A map object, which can be converted to a list or other iterable types.
- Use Cases: Converting data types, performing mathematical operations, and applying transformations.

map Example

```
# Convert Celsius to Fahrenheit
temps_celsius = [0, 20, 30, 40]
temps_fahrenheit = map(lambda c: (c * 9/5) + 32, temps_celsius)

print(list(temps_fahrenheit))
# Output: [32.0, 68.0, 86.0, 104.0]
```

Generator Functions (yield)

- Generators are a type of iterable, like lists, but they produce items lazily (one at a time). This makes them memory-efficient, especially for large datasets.
- `Key Features:`
- Use yield to produce a value and pause execution.
- Can be resumed to continue producing more values.
- Automatically implements the iterator protocol.

Advantages

- Memory Efficient: Stores only the current state, not all values.
- Lazy Evaluation: Generates values only when needed.
- Infinite Sequences: Ideal for generating infinite sequences like Fibonacci numbers or prime numbers.

Comparison to Regular Functions:

- A regular function returns all values at once and terminates.
- A generator function yields values one at a time, preserving memory.

yield Example

```
# Generate Fibonacci sequence up to a limit
def fibonacci(limit):
    a, b = 0, 1
    while a < limit:
        yield a
        a, b = b, a + b

# Using the generator
for num in fibonacci(10):
    print(num)
# Output: 0, 1, 1, 2, 3, 5, 8
```

End of Python Course !