

Lab 4 - Functional Programming (Section 2) - Report

Students:

1. Nguyen Huynh Thao My - ITCSIU21204
 2. Pham Duc Dat - ITITI20184
-

1 Overview

`any()` - is used to test whether any elements of an iterable meet a certain condition.

Syntax: `any()` returns "True" if any element in the iterable is a true value.

`all()` - is used to test whether all elements of an iterable meet a certain condition.

Syntax: `all()` returns "True" if all of the elements are true values.

`Compose` - combines multiple functions into a single function. This combined function applies the given functions in sequence, where the output of one function becomes the input of the next.

`Currying` - We can use higher-order functions to convert a function that takes multiple arguments into a chain of functions that each take a single argument. More specifically, given a function $f(x, y)$, we can define a function g such that $g(x)(y)$ is equivalent to $f(x, y)$. Here, g is a higher-order function that takes in a single argument x and returns another function that takes in a single argument y . This transformation is called currying. Currying can make function composition and partial application more convenient.

2 Results

In this section, you:

- Implemented both traditional and higher-order function methods to generate the square and cube of each element in an array, filter the squared results within a specified range, and identify even numbers in the array.
- Calculated the Euclidean distance between two points using a lambda function.
- Used higher-order functions to extract names of specific animals from a dictionary.
- Manipulated an array of characters with higher-order functions to sum elements from left to right and right to left and concatenate characters with additional elements.

2.1 Exercise 1

You are managing an inventory system for an e-commerce platform. Each product in the inventory has the following properties:

- **name:** The name of the product.
- **price:** The price of the product.
- **stock:** The number of items available in stock.
- **categories:** A list of categories the product belongs to.

The dictionary of products is as below:

```
inventory = [
    {"name": "Laptop", "price": 1200, "stock": 10, "categories": ["electronics", "computers"]},
    {"name": "Smartphone", "price": 800, "stock": 0, "categories": ["electronics", "mobile"]},
    {"name": "Headphones", "price": 150, "stock": 25, "categories": ["electronics", "audio"]},
    {"name": "Desk Chair", "price": 100, "stock": 5, "categories": ["furniture", "office"]},
    {"name": "Notebook", "price": 5, "stock": 100, "categories": ["stationery", "office"]},
]
```

Figure 1: Dictionary of products

Please implement a program/function, by using `map()`, `lambda()`, `filter()` nested inside the usage of functions `any()` and `all()`, that checks the following conditions for the inventory:

- Determine if there are any products that are out of stock.
- Verify if all products in a specific category are available and have a price greater than a specified threshold δ , where δ is input by users.

Code:

Explanation:

The code first checks if there are any products that are out of stock using the `any()`

```
# Check if there are any products that are out of stock
out_of_stock = any(map(lambda x: x['stock'] == 0, products))
print(f"Any product out of stock: {out_of_stock}")
```

Figure 2: Code for checking out-of-stock products

```
# Verify if all products in a specific category are available and have a price greater than a specified threshold  $\delta$ , where  $\delta$  is input by users.
def check_category_availability_and_price(category, delta):
    return all(map(lambda x: x['price'] > delta and x['stock'] > 0, filter(lambda y: category in y['categories'], products)))

category = input("Enter category: ")
delta = float(input("Enter price threshold: "))
category_check = check_category_availability_and_price(category, delta)
print(f"All products in category '{category}' have a price greater than {delta} and are in stock: {category_check}")
```

Figure 3: Code for checking category availability and price

```
question1.py > ...
1 # QUESTION 01
2 products = [
3     {'name': 'Laptop', 'price': 1000, 'stock': 5, 'categories': ['Electronics', 'Computers']},
4     {'name': 'Smartphone', 'price': 800, 'stock': 0, 'categories': ['Electronics', 'Phones']},
5     {'name': 'Book', 'price': 15, 'stock': 10, 'categories': ['Books']},
6     {'name': 'Headphones', 'price': 100, 'stock': 15, 'categories': ['Electronics', 'Audio']},
7 ]
8
9 def main():
10     # Check if there are any products that are out of stock
11     out_of_stock = any(map(lambda x: x['stock'] == 0, products))
12     print(f"Any product out of stock: {out_of_stock}")
13
14     # Verify if all products in a specific category are available and have a price greater than a specified threshold  $\delta$ , where  $\delta$  is
    input by users.
15     def check_category_availability_and_price(category, delta):
16         return all(map(lambda x: x['price'] > delta and x['stock'] > 0, filter(lambda y: category in y['categories'], products)))
17
18     category = input("Enter category: ")
19     delta = float(input("Enter price threshold: "))
20     category_check = check_category_availability_and_price(category, delta)
21     print(f"All products in category '{category}' have a price greater than {delta} and are in stock: {category_check}")
22
23 if __name__ == "__main__":
24     main()
```

Figure 4: Code for main function to allow users input command from the script

```
Any product out of stock: True
Enter category: Headphones
Enter price threshold: 10000
All products in category 'Headphones' have a price greater than 10000.0 and are in stock: True
```

Figure 5: Output of the program

function combined with `map()` and `lambda()` to iterate through the list of products. The result is printed to the console.

Then, the code defines a function `check_category_availability_and_price()` that checks if all products in a specific category are available and have a price greater than a specified threshold δ . This function uses `all()`, `map()`, `filter()`, and `lambda()` to

perform the checks. The function takes the category and the price threshold as input from the user and prints the result to the console.

2.2 Exercise 2

Given a dictionary of users, define curried functions to:

- Filter users by minimum age.
- Format the user's name according to a specified style.
- Calculate the user's score based on a given criteria.
- Combine the curried functions into a data processing pipeline.

The dictionary of users is as below:

```
users = [  
    {"name": "Alice Johnson", "age": 25, "activity": 120},  
    {"name": "Bob Smith", "age": 30, "activity": 150},  
    {"name": "Charlie Brown", "age": 20, "activity": 80},  
    {"name": "Diana Ross", "age": 35, "activity": 200},  
]
```

Figure 6: Dictionary of users

Code:

```
# Filter by minimum age  
def filter_by_min_age(min_age):  
    def inner(users):  
        result = list(filter(lambda user: user['age'] >= min_age, users))  
        print(f"After filtering by minimum age {min_age}: {result}")  
        return result  
    return inner
```

Figure 7: Code for filtering by minimum age

```
# Format name
def format_name(style):
    def inner(users):
        if style == 'uppercase':
            result = list(map(lambda user: {**user, 'name': user['name'].upper()}, users))
        elif style == 'lowercase':
            result = list(map(lambda user: {**user, 'name': user['name'].lower()}, users))
        else:
            result = users
        print(f"After formatting names to {style}: {result}")
        return result
    return inner
```

Figure 8: Code for formatting names

```
# Calculate score
def calculate_score(criteria):
    def inner(users):
        if criteria == 'double':
            result = list(map(lambda user: {**user, 'score': user['score'] * 2}, users))
        else:
            result = users
        print(f"After calculating scores with criteria {criteria}: {result}")
        return result
    return inner
```

Figure 9: Code for calculating scores

Explanation:

The code defines several functions to process a list of users.

First, a ‘compose’ function is defined to combine multiple functions into a single function that applies them in sequence.

Then, three curried functions are defined:

- `filter_by_min_age(min_age)`: Filters users by a minimum age.
- `format_name(style)`: Formats the name of each user to either uppercase or lowercase.
- `calculate_score(criteria)`: Calculates the score of each user based on a given criteria (e.g., doubling the score).

These functions are then combined into a data processing pipeline using the ‘com-

```
def main():
    # Data processing pipeline
    pipeline = compose(
        filter_by_min_age(30),
        format_name('uppercase'),
        calculate_score('double')
    )

    processed_users = pipeline(users)
    print("Final processed users:", processed_users)

if __name__ == "__main__":
    main()
```

Figure 10: Code for main function

```
After calculating scores with criteria double: [{'name': 'Alice', 'age': 30, 'score': 170}, {'name': 'Bob', 'age': 25, 'score': 190}, {'name': 'Charlie', 'age': 35, 'score': 140}]
After formatting names to uppercase: [{'name': 'ALICE', 'age': 30, 'score': 170}, {'name': 'BOB', 'age': 25, 'score': 190}, {'name': 'CHARLIE', 'age': 35, 'score': 140}]
After filtering by minimum age 30: [{'name': 'ALICE', 'age': 30, 'score': 170}, {'name': 'CHARLIE', 'age': 35, 'score': 140}]
Final processed users: [{'name': 'ALICE', 'age': 30, 'score': 170}, {'name': 'CHARLIE', 'age': 35, 'score': 140}]
```

Figure 11: Output of the program

pose' function. The pipeline filters users who are at least 30 years old, formats their names to uppercase, and doubles their scores. The final result is printed to the console.

2.3 Exercise 3

Given a text, define curried functions to:

- Remove punctuation from the text.
- Convert the text to lowercase.
- Split the text into words.
- Filter out common stop words.
- Combine the curried functions into a text processing pipeline.

Code:

```

# Remove punctuation
def remove_punctuation(text):
    return re.sub(r'^\w\s', '', text)

# Convert text to lowercase
def to_lowercase(text):
    return text.lower()

# Split text into words
def split_into_words(text):
    return text.split()

# Filter out common stop words
def filter_stop_words(words):
    stop_words = {'the', 'is', 'in', 'and', 'or', 'of', 'a', 'an'}
    return [word for word in words if word not in stop_words]

# Compose function
def compose(*functions):
    return reduce(lambda f, g: lambda x: f(g(x)), functions, lambda x: x)

```

Figure 12: Code for text processing pipeline for every single problem.

Input: `text = "Hello Thao My, my name is Duc Dat"`

Output: `['hello', 'thao', 'my', 'my', 'name', 'duc', 'dat']`

Another Example:

Input: `text = "Python is great, and so is AI!"`

Output: `['python', 'great', 'so', 'ai']`

Explanation:

The code defines several functions to process a text input.

First, a `compose` function is defined to combine multiple functions into a single function that applies them in sequence.

Then, four functions are defined:

- `remove_punctuation(text)`: Removes punctuation from the text.

```
def main():
    # Build the text processing pipeline
    text_processing_pipeline = compose(
        filter_stop_words,
        split_into_words,
        to_lowercase,
        remove_punctuation
    )

    # Sample text
    text = "Hello Thao My, my name is Duc Dat"

    # Processed text
    processed_text = text_processing_pipeline(text)
    print(processed_text)

if __name__ == "__main__":
    main()
```

Figure 13: Code for main function to allow users input command from the script.

- `to_lowercase(text)`: Converts the text to lowercase.
- `split_into_words(text)`: Splits the text into individual words.
- `filter_stop_words(words)`: Filters out common stop words from the list of words.

These functions are then combined into a text processing pipeline using the `compose` function. The pipeline processes the sample text by removing punctuation, converting it to lowercase, splitting it into words, and filtering out common stop words. The final result is printed to the console.

2.4 Result

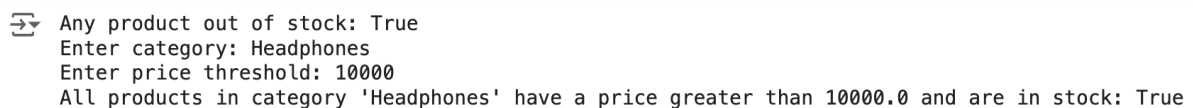
To run the Python code for each exercise, use the following commands in your terminal or command prompt:

- `python3 run_FP.py 1`: This command runs the function for Exercise 1.
- `python3 run_FP.py 2`: This command runs the function for Exercise 2.
- `python3 run_FP.py 3`: This command runs the function for Exercise 3.

After running the commands, you should see the results printed in the terminal. Each command will execute the corresponding main function for the exercise and display the output as described below:

2.4.1 Exercise 1

- Checks if there are any products that are out of stock.
- Verifies if all products in a specific category are available and have a price greater than a specified threshold.

A terminal window showing the output of the first exercise. It starts with a prompt icon, followed by the text: 'Any product out of stock: True', 'Enter category: Headphones', 'Enter price threshold: 10000', and finally 'All products in category 'Headphones' have a price greater than 10000.0 and are in stock: True'.

```
Any product out of stock: True
Enter category: Headphones
Enter price threshold: 10000
All products in category 'Headphones' have a price greater than 10000.0 and are in stock: True
```

Figure 14: Output of the Exercise 1

2.4.2 Exercise 2

- Filters users by minimum age.
- Formats the user's name according to a specified style.
- Calculates the user's score based on a given criteria.
- Combines the curried functions into a data processing pipeline.

2.4.3 Exercise 3

- Removes punctuation from the text.

```

After calculating scores with criteria double: [{'name': 'Alice', 'age': 30, 'score': 170}, {'name': 'Bob', 'age': 25, 'score': 190}, {'name': 'Charlie', 'age': 35, 'score': 140}]
After formatting names to uppercase: [{'name': 'ALICE', 'age': 30, 'score': 170}, {'name': 'BOB', 'age': 25, 'score': 190}, {'name': 'CHARLIE', 'age': 35, 'score': 140}]
After filtering by minimum age 30: [{'name': 'ALICE', 'age': 30, 'score': 170}, {'name': 'CHARLIE', 'age': 35, 'score': 140}]
Final processed users: [{'name': 'ALICE', 'age': 30, 'score': 170}, {'name': 'CHARLIE', 'age': 35, 'score': 140}]

```

Figure 15: Output of the Exercise 2

- Converts the text to lowercase.
- Splits the text into words.
- Filters out common stop words.
- Combines the curried functions into a text processing pipeline.

```

• (base) → Lab04 python3 run_FP.py 3
  ['hello', 'thao', 'my', 'my', 'name', 'duc', 'dat']

```

Figure 16: Output of the Exercise 3