



School of Computer Science and Artificial Intelligence

Lab Assignment-9.1

Course Name : **AI Assistant Coding**
Name of the Student : **Shaik Naved Ahmed**
Registration No : **2303A54053**
Batch No : **47-B**

Problem 1: Consider the following Python function:

```
def find_max(numbers):  
    return max(numbers)
```

Task:

- Write documentation for the function in all three formats:
 - (a) Docstring
 - (b) Inline comments
 - (c) Google-style documentation
- Critically compare the three approaches. Discuss the advantages, disadvantages, and suitable use cases of each style.
- Recommend which documentation style is most effective for a mathematical utilities library and justify your answer.

SCREENSHOT OF GENERATED CODE:

```
lab9.1.py X  
1 #Docstring Documentation with complete code  
2 def find_max(numbers):  
3     """  
4     This function takes a list of numbers as input and returns the maximum value from the list.  
5     """  
6     Parameters:  
7     numbers (list): A list of numerical values.  
8     Returns:  
9     The maximum value from the input list.  
10    # Check if the input list is empty  
11    if not numbers:  
12        raise ValueError("The input list cannot be empty.")  
13    # Use the built-in max function to find the maximum value in the list  
14    return max(numbers)  
15  
16 #Example usage  
17 numbers = [3, 1, 4, 1, 5, 9]  
18 max_value = find_max(numbers)  
19 print(f"The maximum value in the list is: {max_value}")  
20  
21 #Inline Documentation with complete code  
22 def find_max(numbers):  
23     """  
24     This function takes a list of numbers as input and returns the maximum value from the list.  
25     """  
26     Parameters:  
27     numbers (list): A list of numerical values.  
28     Returns:  
29     The maximum value from the input list.  
30     # Check if the input list is empty  
31     if not numbers:  
32         raise ValueError("The input list cannot be empty.")  
33     # Use the built-in max function to find the maximum value in the list  
34     return max(numbers)  
35  
36 #Example usage  
37 numbers = [3, 1, 4, 1, 5, 9]  
38 max_value = find_max(numbers)  
39 print(f"The maximum value in the list is: {max_value}")  
40  
41 #Google-Style Documentation with complete code  
42 def find_max(numbers):  
43     """  
44     Finds the maximum value in a list of numbers.  
45     """  
46     Args:  
47     numbers (list): A list of numerical values. The list should not be empty.  
48     Returns:  
49     The maximum value from the input list.  
50     Raises:  
51     ValueError: If the input list is empty.  
52     """  
53     if not numbers:  
54         raise ValueError("The input list cannot be empty.")  
55     return max(numbers)  
56  
57 #Example usage  
58 numbers = [3, 1, 4, 1, 5, 9]  
59 max_value = find_max(numbers)  
60 print(f"The maximum value in the list is: {max_value}")  
61  
62 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
63 (base) navedahmedshaikh@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/naivedahmedshaikh/Documents/3-2/AIAC/lab.9.1.py  
64 Zsh  
65 (base) navedahmedshaikh@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/naivedahmedshaikh/Documents/3-2/AIAC/lab.9.1.py  
66 The maximum value in the list is: 9  
67 The maximum value in the list is: 9  
68 The maximum value in the list is: 9  
69 (base) navedahmedshaikh@Naveds-MacBook-Air AIAC %
```

EXPLANATION:

The function `find_max` takes a list of numbers as input and returns the maximum value from that list. It first checks if the input list is empty and raises a `ValueError` if it is. If the list is not empty, it uses the built-in `max` function to find and return the maximum value. The function is documented using three different styles: docstring documentation, inline documentation, and Google-style documentation, providing clear explanations of the function's purpose, parameters, return value, and potential exceptions.

Style	Advantages	Disadvantages	Suitable Use Cases
Docstring (standard)	Simple, readable, built into Python help tools	Less structured than formal styles	Small scripts, educational code
Inline comments	Explain logic step-by-step	Can clutter code, not accessible via <code>help()</code>	Complex algorithms needing explanation
Google-style	Highly structured, professional, IDE-friendly, good for teams	Slightly longer to write	Libraries, APIs, production code

Problem 2: Consider the following Python function:

```
def login(user, password, credentials):  
    return credentials.get(user) == password
```

Task:

1. Write documentation in all three formats.
2. Critically compare the approaches.
3. Recommend which style would be most helpful for new developers onboarding a project, and justify your choice.

SCREENSHOT OF GENERATED CODE:

```
72 #task2  
73  
74 def login(user, password, credentials):  
75     return credentials.get(user) == password  
76  
77 #docstring documentation with complete code  
78 def login(user, password, credentials):  
79     """  
80     This function checks if the provided username and password match the credentials stored in a dictionary.  
81  
82     Parameters:  
83     user (str): The username to be checked.  
84     password (str): The password to be checked.  
85     credentials (dict): A dictionary containing username-password pairs.  
86  
87     Returns:  
88     bool: True if the username and password match the credentials, False otherwise.  
89     """  
90     return credentials.get(user) == password  
91  
92 #Example usage  
93 credentials = {'user1': 'pass1', 'user2': 'pass2'}  
94 print(login('user1', 'pass1', credentials)) # Output: True  
95 print(login('user1', 'wrongpass', credentials)) # Output: False  
96  
97 #Inline Documentation with complete code  
98 def login(user, password, credentials):  
99     """  
100     This function checks if the provided username and password match the credentials stored in a dictionary.  
101  
102     Parameters:  
103     user (str): The username to be checked.  
104     password (str): The password to be checked.  
105     credentials (dict): A dictionary containing username-password pairs.  
106  
107     Returns:  
108     bool: True if the username and password match the credentials, False otherwise.  
109     """  
110     # Use the get method to retrieve the password for the given user and compare it with the provided password  
111     return credentials.get(user) == password  
112  
113 #Example usage  
114 credentials = {'user1': 'pass1', 'user2': 'pass2'}  
115 print(login('user1', 'pass1', credentials)) # Output: True  
116 print(login('user1', 'wrongpass', credentials)) # Output: False  
117  
118 #Google-Style Documentation with complete code  
119 def login(user, password, credentials):  
120     """  
121     Checks if the provided username and password match the credentials stored in a dictionary.  
122  
123     Args:  
124     user (str): The username to be checked.  
125     password (str): The password to be checked.  
126     credentials (dict): A dictionary containing username-password pairs. The keys are usernames and the values are passwords.  
127  
128     Returns:  
129     bool: True if the username and password match the credentials, False otherwise.  
130     """  
131     return credentials.get(user) == password  
132  
133 #Example usage  
134 credentials = {'user1': 'pass1', 'user2': 'pass2'}  
135 print(login('user1', 'pass1', credentials)) # Output: True  
136 print(login('user1', 'wrongpass', credentials)) # Output: False
```

EXPLANATION:

The function `login` takes three parameters: `user`, `password`, and `credentials`. It checks if the provided username and password match the credentials stored in a dictionary. The function uses the `get` method of the dictionary to retrieve the password associated with the given username and compares it with the provided password. If they match, it returns `True`; otherwise, it returns `False`. The function is documented using three different styles: docstring

documentation, inline documentation, and Google-style documentation, providing clear explanations of the function's purpose, parameters, return value, and potential exceptions.

Style	Advantages	Disadvantages	Best Use Cases
Docstring (standard)	Simple, readable, built into Python help tools	Less structured for large projects	Small scripts, student projects
Inline comments	Explain logic directly near code	Not visible via help(), can clutter code	Complex internal logic explanation
Google-style	Structured, professional, IDE-friendly, great for teams	Slightly longer to write	Libraries, APIs, collaborative projects

Problem 3: Calculator (Automatic Documentation Generation)

Task: Design a Python module named calculator.py and demonstrate automatic documentation generation.

Instructions:

- Create a Python module calculator.py that includes the following functions, each written with appropriate docstrings:
 - o add(a, b) – returns the sum of two numbers
 - o subtract(a, b) – returns the difference of two numbers
 - o multiply(a, b) – returns the product of two numbers
 - o divide(a, b) – returns the quotient of two numbers
- Display the module documentation in the terminal using Python's documentation tools.
- Generate and export the module documentation in HTML format using the pydoc utility, and open the generated HTML file in a web browser to verify the output.

SCREENSHOT OF GENERATED CODE:

The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'AIAC' with various files including 'calculator.py'. The code editor displays the content of 'calculator.py', which includes a docstring for the module and four functions: add, subtract, multiply, and divide. The terminal shows the output of the 'pydoc' command, displaying the module documentation in a structured format.

```
1 #Lab 9.1
2 #Problem-3
3
4
5 calculator.py - Simple calculator module with basic operations.
6
7
8 def add(a, b):
9     """Return the sum of a and b."""
10    return a + b
11
12 def subtract(a, b):
13     """Return the difference of a and b."""
14    return a - b
15
16 def multiply(a, b):
17     """Return the product of a and b."""
18    return a * b
19
20 def divide(a, b):
21     """
22     Return the quotient of a and b.
23
24     Raises:
25         ValueError: If b is zero.
26     """
27     if b == 0:
28         raise ValueError("Cannot divide by zero")
29    return a / b
30
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Help on module calculator:

NAME

calculator - calculator.py - Simple calculator module with basic operations.

FUNCTIONS

add(a, b)

Return the sum of a and b.

divide(a, b)

Return the quotient of a and b.

Raises:

ValueError: If b is zero.

multiply(a, b)

Return the product of a and b.

subtract(a, b)

Return the difference of a and b.

FILE

/Users/navedahmedshah/Documents/3-2/AIAC/calculator.py

Help on calculator line 1/24 (END) (press h for help or q to quit)

calculator </Users/navedahmedshaik/Documents/3-2/AIAC/calculator.py>.

calculator.py - Simple calculator module with basic operations.

Functions

add(a, b)

Return the sum of a and b.

divide(a, b)

Return the quotient of a and b.

Raises:

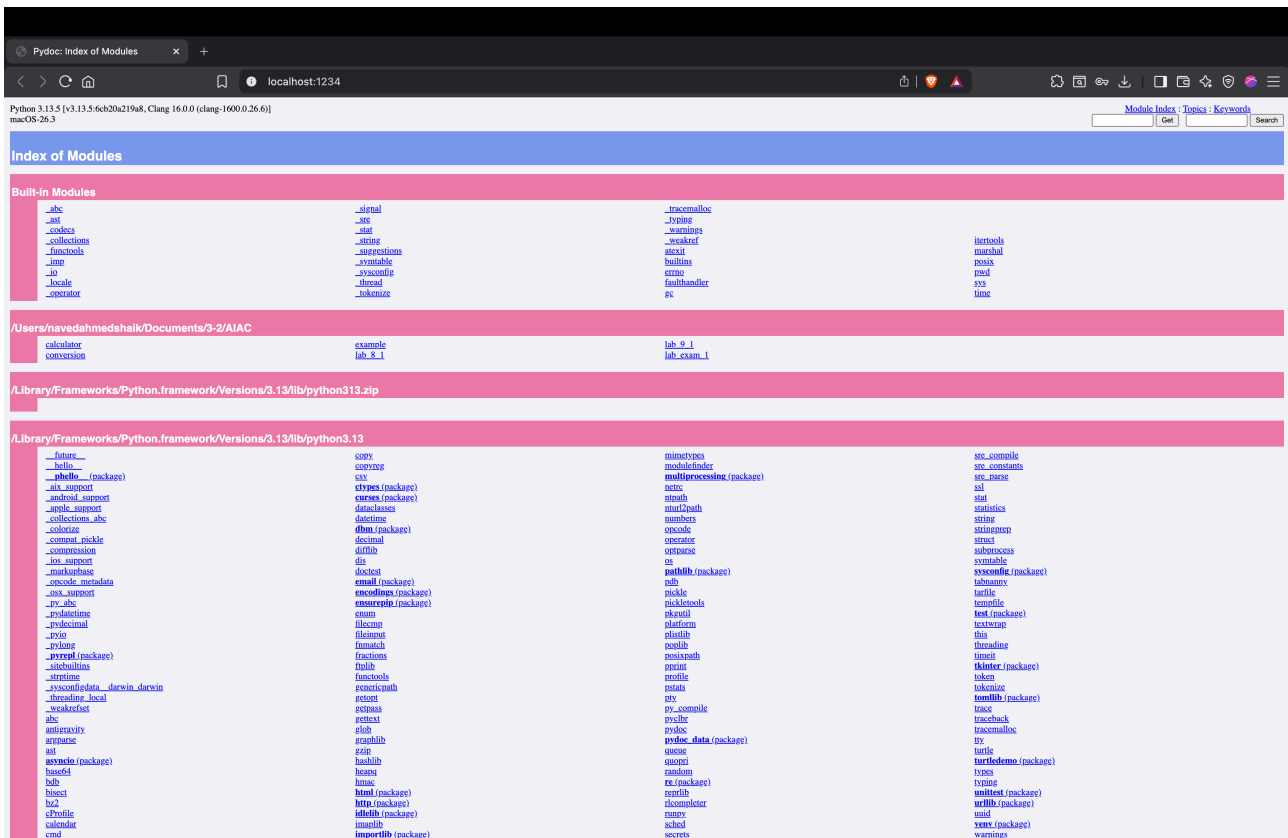
ValueError: If b is zero.

multiply(a, b)

Return the product of a and b.

subtract(a, b)

Return the difference of a and b.

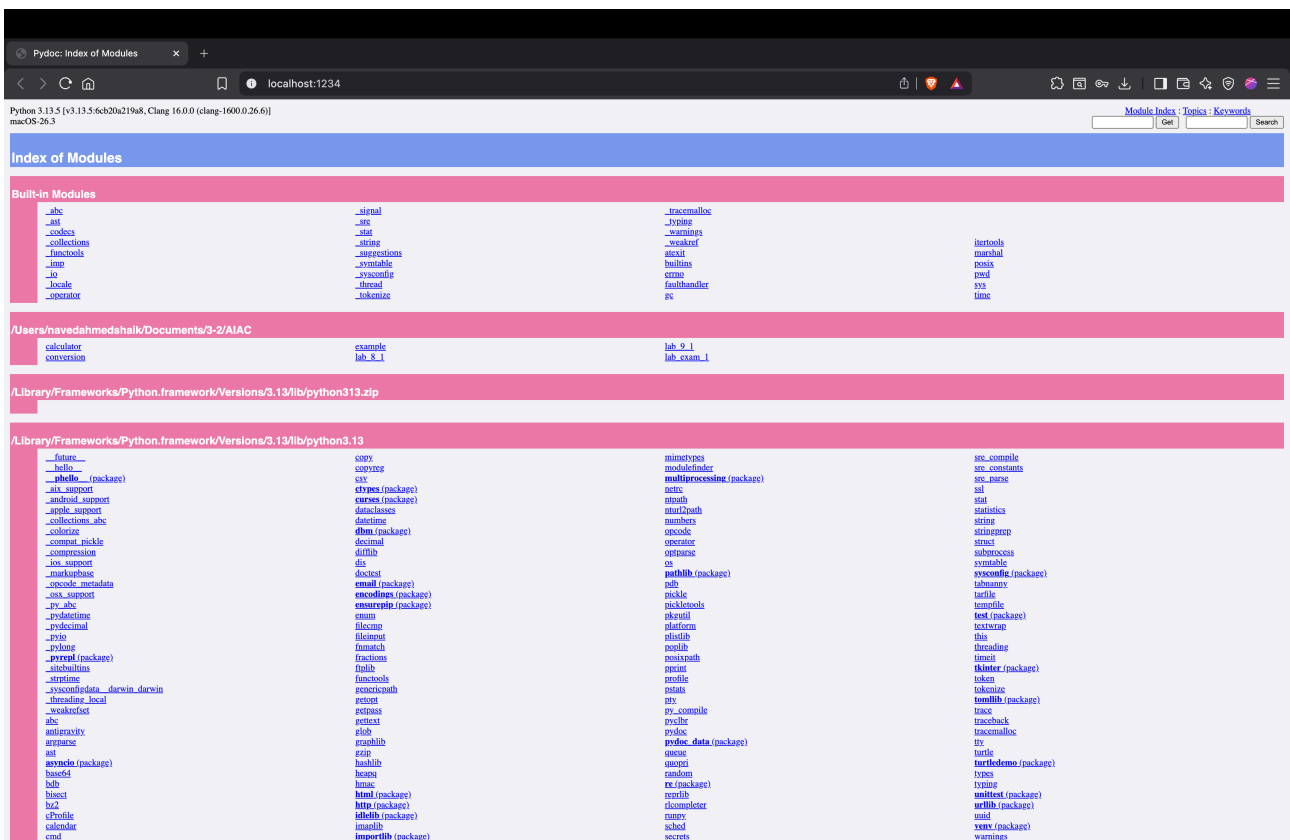
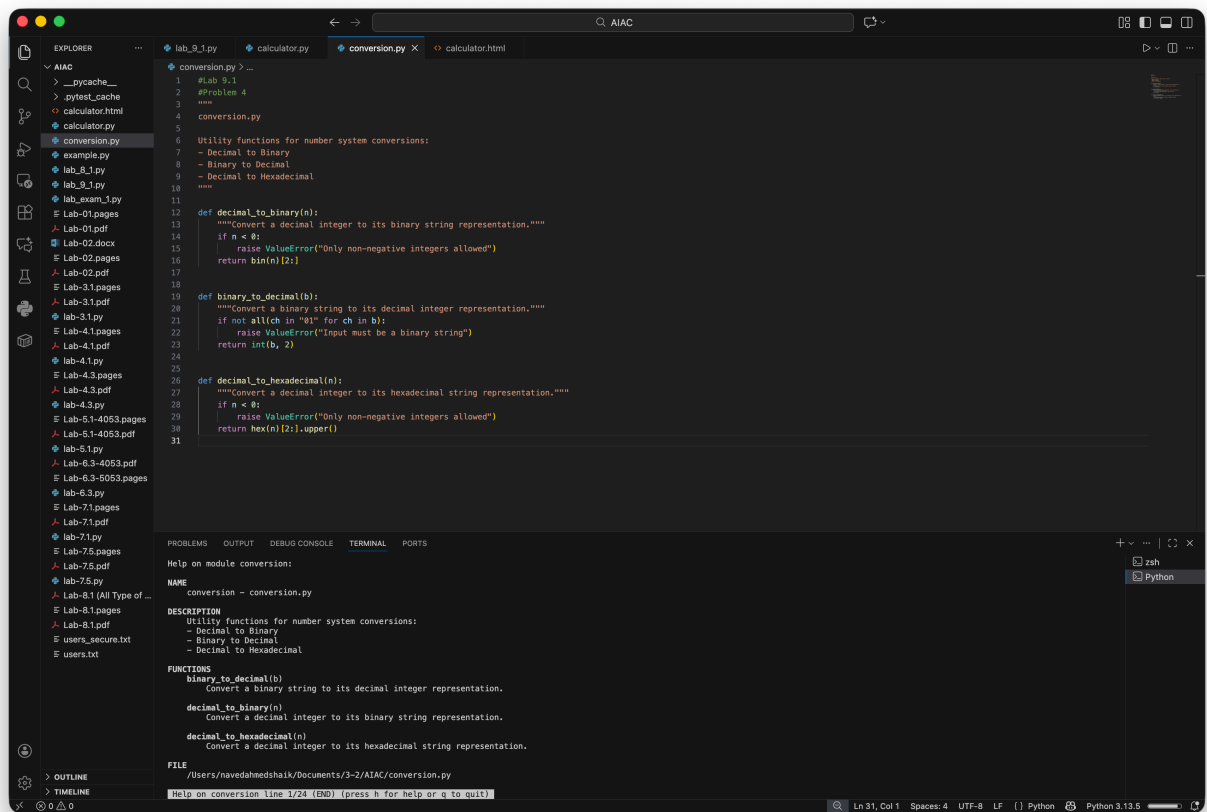


Problem 4: Conversion Utilities Module

Task:

1. Write a module named `conversion.py` with functions:
 - o `decimal_to_binary(n)`
 - o `binary_to_decimal(b)`
 - o `decimal_to_hexadecimal(n)`
2. Use Copilot for auto-generating docstrings.
3. Generate documentation in the terminal.
4. Export the documentation in HTML format and open it in a browser.

SCREENSHOT OF GENERATED CODE:



[index](#)
conversion </Users/navedahmedshaik/Documents/3-2/AIAC/conversion.py>

conversion.py

Utility functions for number system conversions:

- Decimal to Binary
- Binary to Decimal
- Decimal to Hexadecimal

Functions

binary_to_decimal(b)
Convert a binary string to its decimal integer representation.

decimal_to_binary(n)
Convert a decimal integer to its binary string representation.

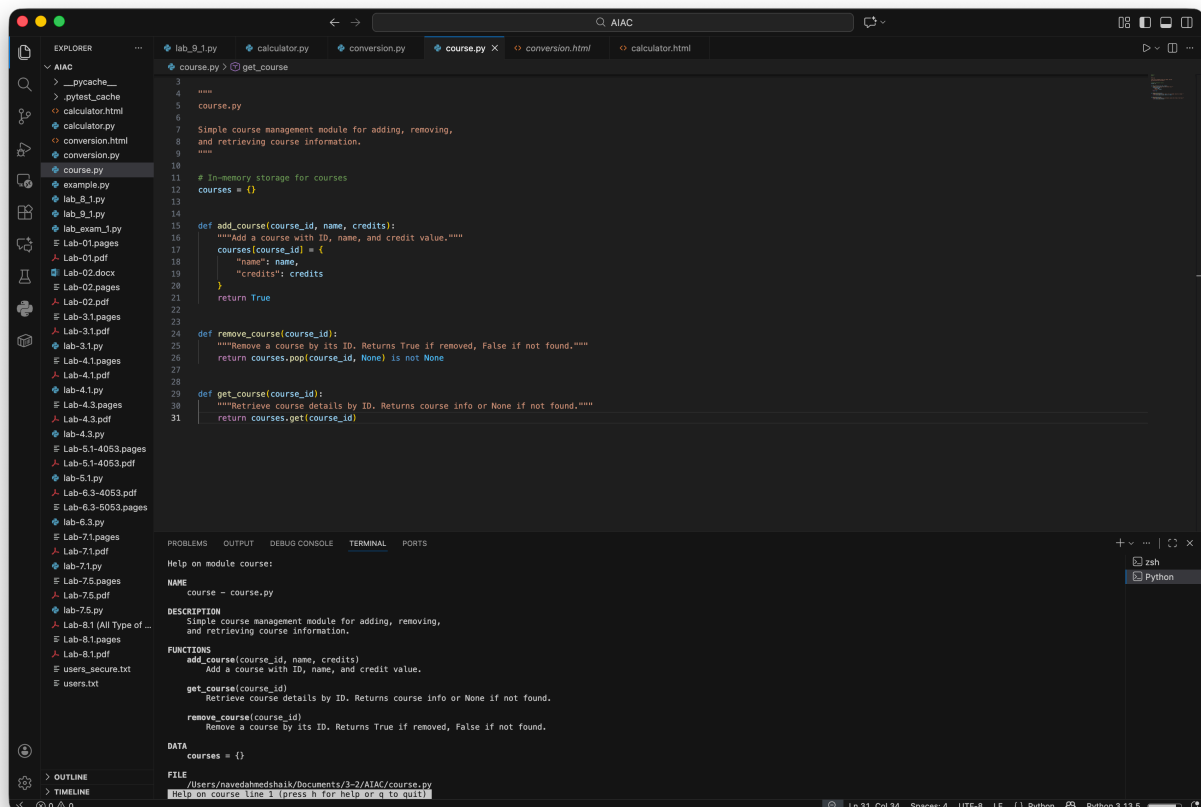
decimal_to_hexadecimal(n)
Convert a decimal integer to its hexadecimal string representation.

Problem 5 – Course Management Module

Task:

1. Create a module `course.py` with functions:
 - o `add_course(course_id, name, credits)`
 - o `remove_course(course_id)`
 - o `get_course(course_id)`
2. Add docstrings with Copilot.
3. Generate documentation in the terminal.
4. Export the documentation in HTML format and open it in a browser.

SCREENSHOT OF GENERATED CODE:



The screenshot displays a web browser window with the address bar showing 'localhost:1234'. The page is titled 'Index of Modules' and is a Pydoc-generated index. The content is organized into sections: 'Built-in Modules', 'Users/navedahmedshah/Docs/2-2/AIAC', and 'Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13.zip'. Each section contains a list of modules and their sub-modules, such as 'abc', 'ast', 'collections', 'functools', 'itertools', 'math', 'random', 're', 'urllib', 'xml', 'yaml', 'zipfile', etc. The page is styled with a light blue header and a light blue background.