DS&AI

Pythonprogramming

Practice Sheet 02 Function & Modules

Q1 Which of the below container is used to encapsulate multiple dictionaries into single list of dictionaries?

(A) defaultdict (B) Deque

(C) ChainMap (D) Ordereddict

Q2 The output of below Python Code Segment is

```
_____from collections import OrderedDict
d = OrderedDict()
d['p'] = 1
d['q'] = 2
d['r'] = 3
d['s'] = 4
d.pop('q')
d.pop('s')
d['s'] = 2
d['t'] = 5
d.pop('r')
d['q'] = 1
for key, value in d.items():
print(key, value,end=' ')
```

(A) p 1 s 4 t 5 q 1

(B) p 1 s 2 t 5 q 1

(C) p 1 s 2 t 5 q 2

(D) p 1 s 4 t 5 q 2

Q3 Which of the below class can be used to implement both stack and queue?

(A) chainMap (B) lifoQueue

(C) Ordereddict (D) deque

Q4 Arrange the below in the order of Subset to Superset relativity.

1. Module 2. Class 3. Package 4. Method

(A) 3, 2, 1, 4 (B) 4, 2, 3, 1

(C) 4, 2, 1, 3 (D) 3, 1, 2, 4

Q5 Identify True Statement(s) from below.

(A) Arrays as Lists are Slower Compared to Arrays as numpy object

(B) defaultdict preserve the order of insertion of elements

(C) Dictionaries if implemented using defaultdict class, it never raises a Key Error

(D) A ChainMap is a class that accumulates multiple dictionaries into a single list.

Q6 The return value of below function fun(5,1,9) is

_____

```
def fun(x,y,z):
  if x==y or y==z or x==z:
    return x-y+z
  elif x>y:
    return x+fun(x-1,y,z+2)
  elif y>z:
    return y+fun(x+1,y-1,z)
  else:
    return z+fun(x,y,z-1)
```

Q7 Which of the below statements raise a syntax error?

(A) max = lambda a, b : x if(a > b)

(B) square = lambda x : x*x if(x > 0) else None

(C) calc = lambda num: "Even number" if num % 2 == 0 else "Odd number"

(D) result = lambda a,b=4,c=2: a+b+c

Q8 The output of below code segment is _____

```
def f(x):
  if x>20:
    return x-1
  else:
    return x+f(f(x+2))
print(f(10))
```

Answer Key

Q1	(C)	Q5	(A, C, D)
Q2	(B)	Q6	31
Q3	(D)	Q7	(A)
Q4	(C)	Q8	105

Hints & Solutions

Q1 Text Solution:

`ChainMap` is used to group multiple dictionaries together, allowing you to treat them as a single mapping.

Q2 Text Solution:

1. An `OrderedDict` is created, and the following key-value pairs are added:
   - 'p': 1
   - 'q': 2
   - 'r': 3
   - 's': 4
2. The entries for 'q' and 's' are popped:
   - After `d.pop('q')`, 'q' is removed.
   - After `d.pop('s')`, 's' is removed.
   - The current state of the dictionary is: `OrderedDict([('p', 1), ('r', 3)])`
3. The entry for 's' is added back with a new value:
   - `d['s'] = 2`
   - The current state is: `OrderedDict([('p', 1), ('r', 3), ('s', 2)])`
4. The entry for 'r' is popped:
   - After `d.pop('r')`, 'r' is removed.
   - The current state is: `OrderedDict([('p', 1), ('s', 2)])`
5. Finally, 'q' is added back with a new value:
   - `d['q'] = 1`
   - The final state of the dictionary is: `OrderedDict([('p', 1), ('s', 2), ('t', 5), ('q', 1)])`
6. The loop prints each key-value pair in order:
   - The output will be: `p 1 s 2 t 5 q 1`

So, the correct answer is:
B. p 1 s 2 t 5 q 1

Q3 Text Solution:

The `deque` (double-ended queue) class from the `collections` module can be used to implement both stacks and queues, allowing you to append and pop elements from both ends efficiently.

Q4 Text Solution:

1. **Method** - a function defined within a class (the smallest).
2. **Class** - a blueprint for creating objects, which can contain methods.
3. **Module** - a file containing Python code (which can include classes and methods).
4. **Package** - a collection of modules (the largest).

So, the correct order is:
C. 4, 2, 1, 3 (Method, Class, Module, Package)

Q5 Text Solution:

A. Arrays as Lists are Slower Compared to Arrays as numpy object
True. NumPy arrays are optimized for performance and allow for faster operations compared to Python lists, especially for large datasets.
B. defaultdict preserve the order of insertion of elements
True. `defaultdict`, like regular dictionaries (since Python 3.7), preserves the order of insertion.
C. Dictionaries if implemented using defaultdict class, it never raises a Key Error
True. A `defaultdict` provides a default value for missing keys, so it will not raise a `KeyError`.
D. A ChainMap is a class that accumulates multiple dictionaries into a single list.
False. A `ChainMap` does not create a list; it groups multiple dictionaries together to act as one, but they remain separate dictionaries.
Based on this analysis, the true statements are:
A, B, C.

Q6 Text Solution:

1. **Input:** `fun(5, 1, 9)`
   - Since x (5) is not equal to y (1) or z (9), and y (1) is not equal to z (9), the first condition is false.
   - The second condition `x > y` (5 > 1) is true, so it will execute `return 5 + fun(4, 1, 11)`.
2. **Next Call:** `fun(4, 1, 11)`
   - Again, none of the first conditions are true, and `x > y` (4 > 1) is true.
   - It will execute `return 4 + fun(3, 1, 13)`.
3. **Next Call:** `fun(3, 1, 13)`
   - None of the first conditions are true, and `x > y` (3 > 1) is true.
   - It will execute `return 3 + fun(2, 1, 15)`.
4. **Next Call:** `fun(2, 1, 15)`
   - None of the first conditions are true, and `x > y` (2 > 1) is true.
   - It will execute `return 2 + fun(1, 1, 17)`.
5. **Next Call:** `fun(1, 1, 17)`
   - Now, `x == y` (1 == 1), so it will execute `return 1 - 1 + 17`, which is `0 + 17`, resulting in 17.

Now, we can backtrack through the recursive calls:

- `fun(2, 1, 15)` returns `2 + 17 = 19`
- `fun(3, 1, 13)` returns `3 + 19 = 22`
- `fun(4, 1, 11)` returns `4 + 22 = 26`
- `fun(5, 1, 9)` returns `5 + 26 = 31`

Thus, the return value of `fun(5, 1, 9)` is **31**.

Q7 **Text Solution:**
result = lambda a, b=4, c=2: a+b+c
Correct A

Q8 **Text Solution:**
Initial Call:

- **Call:** `f(10)`
  - Since `10` is not greater than `20`, we go to the `else` branch: `return 10 + f(f(12))`.

Next Calls:

1. **Call:** `f(12)`
   - Not greater than `20`, so: `return 12 + f(f(14))`.
2. **Call:** `f(14)`
   - Not greater than `20`, so: `return 14 + f(f(16))`.
3. **Call:** `f(16)`
   - Not greater than `20`, so: `return 16 + f(f(18))`.
4. **Call:** `f(18)`
   - Not greater than `20`, so: `return 18 + f(f(20))`.
5. **Call:** `f(20)`
   - Not greater than `20`, so: `return 20 + f(f(22))`.
6. **Call:** `f(22)`
   - Now greater than `20`, so: `return 22 - 1`, which is `21`.

Backtracking:
Now, we can substitute back:

- **For `f(20)`**:
  Returns `20 + f(21)`, where `f(21)` is calculated next.

7. **Call:** `f(21)`
   - Greater than `20`, so: `return 21 - 1`, which is `20`.

Substituting back:

- **For `f(20)`**: `20 + 20 = 40`.

Now, substituting back through the previous calls:

- **For `f(18)`**:
  Returns `18 + f(40)`.

8. **Call:** `f(40)`
   - Greater than 20, so: `return 40 - 1`, which is 39.

Now, substituting back:

- **For `f(18)`**: `18 + 39 = 57`.

Continuing to backtrack:

- **For `f(16)`**:
  Returns `16 + f(57)`.

9. **Call:** `f(57)`
   - Greater than 20, so: `return 57 - 1`, which is 56.

Continuing:

- **For `f(16)**: 16 + 56 = 72`.

Next:

- **For `f(14)**: Returns` 14 + f(72)`.

10. **Call:** `f(72)`
    - Greater than 20, so: `return 72 - 1`, which is 71.

Continuing:

- **For `f(14)**: 14 + 71 = 85`.

Next:

- **For `f(12)**: Returns` 12 + f(85)`.

11. **Call:** `f(85)`
    - Greater than 20, so: `return 85 - 1`, which is 84.

Continuing:

- **For `f(12)**: 12 + 84 = 96`.

Finally:

- For `f(10)**: Returns` 10 + f(96)`.

12. **Call:** `f(96)`
    - Greater than 20, so: `return 96 - 1`, which is 95.

So, for the final calculation:

- **For `f(10)**: 10 + 95 = 105`.