Week 11 - Graded Theory

Problem 1

Question

What will be the output of the following code snippet? [MCQ]

```
try:
    if '1' != 1:
        raise "custom error"
    else:
        print("No error occurred")
except "custom error":
    print ("Caution: An error occurred")
```

(a) prints `Caution: An error occurred`

(b) prints `No error occurred`

(c) Program ends with an error

(d) None of these

Answer

(c)

Solution

The exception "custom error" is invalid. It is not caught by the `except` block as it is neither a system defined exception nor user defined exception. Hence, an error is shown to the user.

Problem 2

Question

Given variables `a` and `b` are already defined. Select the true statements. [MSQ]

```
try:
    c = a % b
    if c == 0:
        raise ValueError
except ValueError:
    print('exception')
finally:
    print('finally')
```

(a) Given `a = 10, b = 6`, the code prints

```
finally
```

(b) Given `a = 10, b = 6`, the code prints

```
exception
finally
```

(c) Given `a = 10, b = 5`, the code prints

```
finally
```

(d) Given `a = 10, b = 5`, the code prints

```
exception
finally
```

Answer

(a), (d)

Solution

If `c` is not equal to 0, only `finally` is printed due to the mandatory execution of `finally` block. When `c` is equal to zero, a `ValueError` exception occurs. This prints `exception` and `finally` both.

Problem 3

Question

Given `a = 10, b = 0`. What exception occurs in the below code snippet? [MCQ]

```
try:
    c = a % b
    if c == 0:
        raise ValueError
except ValueError:
    print('exception')
```

(a) `ValueError`

(b) `ZeroDivisionError`

(c) `NameError`

(d) None of these

Answer

(b)

Solution

Here, the denominator in the division is zero (`b = 0`). This causes the program to end abruptly, raising `ZeroDivisionError`.

Question 4

```
iterable_obj = iter(L)
while True:
    try:
        item = int(next(iterable_obj))
        print(item)
    except TypeError:
        print('Error1')
        break
    except ValueError:
        print('Error2')
        break
    except:
        print('Error3')
        break
```

Suppose `L` is a non-empty list of elements. Which of the following statements are true for the given code?[MSQ]

(a) `Error1` will be printed, if element of the list is not convertible to `int`.

(b) `Error2` will be printed, if element of the list is not convertible to `int`.

(c) `Error3` will be printed when the iteration for `iterable_obj` will over.

(d) `Error3` will be printed, if element of the list is not convertible to `int`.

(e) `Error2` will never be printed.

Answer

(a), (b), (c)

Solution

Option (a) is correct if the list `L` has another list or tuple inside as an element then `TypeError` will occur during conversion into `int` and `Error1` will be printed.

Option (b) is correct if list `L` has a string element like `'a'` or `'1.5'` then `ValueError` will occur during conversion into `int` and `Error2` will be printed.

Option (c) is correct because after completion of all iterations when we use `next` again then `StopIteration` error occurs then `Error3` will be printed according to code.

Problem 5

Question

```
L = [y - x for x in [1, 2, 3] for y in [3, 4, 5] if y > x]
```

Which of the following codes are equivalent to the above code? [MSQ]

(a)

```
L = []
for x in [1, 2, 3]:
    for y in [3, 4, 5]:
        if y > x:
            L.append(y - x)
```

(b)

```
L = []
for y in [3, 4, 5]:
    for x in [1, 2, 3]:
        if y > x:
            L.append(y - x)
```

(c)

```
L = []
for x in [1, 2, 3]:
    for y in [3, 4, 5]:
        if y > x:
            L += [y - x]
```

(d)

```
L = []
for y in [3, 4, 5]:
    for x in [1, 2, 3]:
        if y > x:
            L += [y -x]
```

Answer

(a), (c)

Solution

Option (a) and (c) are equivalent codes of given list comprehension. Execute all codes in python IDE for verifying.

Problem 6

Question

```
l1 = []
for i in range(len(L)):
    s = 0
    for j in range(len(L[i])):
        s += L[i][j]
    l1.append(s)
```

Let `L` is a list of the lists that hold positive integers which are already initialized. Which of the following codes are equivalent to the above code? [MSQ]

(a)

```
l1 = [sum(L[i]) for i in range(len(L))]
```

(b)

```
l1 = [sum(L[i]) for i in range(len(L)) for j in range(len(L))]
```

(c)

```
l1 = [sum(L[j]) for i in range(1) for j in L]
```

(d)

```
l1 = [sum(j) for i in range(1) for j in L]
```

Answer

(a), (d)

Solution

Option (a) and (d) are equivalent list comprehension codes of above given code. Execute all codes in python IDE to observe the code and verify the answer.

Problem 7

Question

`L1 = [0, 1, 2, 3]` and `L2 = [0, 1, 4, 9]` are two lists. Which of the following code snippets print below output. The value at each line is the sum of elements from each list at the respective position. [MSQ]

```
0
2
6
12
```

(a)

```
for i, j in zip(L1, L2):
    print(i+j)
```

(b)

```
for i, j in enumerate(L2):
    print(i+j)
```

(c)

```
f = lambda i: i ** 2
for i in L1:
    print(i+f(i))
```

(d) None of these

Answer

(a), (b), (c)

Solution

- `zip` function creates a list of tuples from items of the two lists. The values of each tuple are added and printed on a new line.
- `enumerate` function gives a list of tuples where the first value is the index and the second value is the item of the actual list which is `L2`. The element of the first list `L1` is actually the index position of the second list `L2`. The values of each tuple are added and printed on a new line.
- Lambda function takes a number `i` and returns the square of this number. The second list `L2` is essentially the squares of elements from the first list `L1`. Each value `i` of the list `L1` is added to the `f(i)` which is equivalent to the corresponding element in `L2`. The sum is printed on a new line.

Problem 8

Question

We wish to print all the elements of a list `course_list` on separate lines without any error. Which of the following are valid ways to accomplish this? [MSQ]

```
course_list = ['Python', 'Java', 'PDSA']
def list_items(course_list):
    for item in course_list:
        yield item

list_gen = list_items(course_list)
```

(a)

```
for item in course_list:
    print(item)
```

(b)

```
print(next(list_gen))
print(next(list_gen))
print(next(list_gen))
```

(c)

```
for item in list_gen:
    print(item)
```

(d)

```
print(next(list_gen), next(list_gen),next(list_gen), sep="\n")
```

Answer

(a), (b), (c), (d)

Solution

All options are valid ways to print elements of a list. Option (b), (c), and (d) uses generator function `list_items()` to retrieve elements of the list `course_list` using `next()`.

Problem 9

Question

```
marks = [50, 30, 70, 60, 80, 90, 15, 45, 65]

# fill code

print(result)
```

Which of the following codes are correct to fill in the above code (`# fill code`) so that it prints the list of marks that are greater than 60? [MSQ]

(a)

```
def high_marks(n):
    if n > 60:
        return True
result = list(filter(high_marks, marks))
```

(b)

```
result = list(filter(lambda n : (n > 60), marks))
```

(c)

```
result = list((lambda n : (n > 60), marks))
```

(d)

```
result = [i for i in marks if i > 60 ]
```

(e)

```
result = [i if i > 60 for i in marks ]
```

Answer

(a), (b), (d)

Solution

In option (a) using `filter` , each value of `marks` list will be passed to function `high_marks` as a parameter and those values which are greater than 60, will return into resultant list.

In option (b) using `filter` , each value of `marks` list will be passed to `lambda` function and those values which are greater than 60, will return into the resultant list.

In option (d) using `list comprehension` those values which are greater than 60, will return into the resultant list.

Problem 10

Question

Suppose `salary_list` is a list of employee's salaries (salary is a `float` value). Which of the following codes print the list of salaries after incremented by 10% in each salary? [MSQ]

(a)

```
S = [i + 10 * i for i in salary_list]
print(S)
```

(b)

```
S = [i + .1 * i for i in salary_list]
print(S)
```

(c)

```
S = list(map(lambda n: n + .1 * n, salary_list))
print(S)
```

(d)

```
def inc(n):
    return n + .1 * n
S = list(map(inc, salary_list))
print(S)
```

(e)

```
S = []
for i in salary_list:
    S.append(i + .1 * i)
print(S)
```

Answer

(b), (c), (d), (e)

Solution

option (a) is incorrect, it reads each value of `salary_list` and adds 10 times of it to each value and assigns this new list to `S`.

In option (b), using `list comprehension` increments each value of `salary_list` by 10 % and assigns this new list to `S`. In option (c), `map` goes through each value of `salary_list` and apply `lambda` function to increment it by 10 %, the resulting list is assigned to `S`. In option (d), `map` goes through each value of `salary_list` and apply `inc` function to increment it by 10 %, the resulting list is assigned to `S`. In option (e), all values of `salary_list` are accessed using a loop, incremented by 10% and appended to the list `S`.