Unit - 4 MCQs

1. Arithmetic operators that cannot be used with strings are the subtraction operator (-) and the division operator (/).
2. The function used to find the length of a string is `len()`.
3. The output of the Python code `str1 = "6/4"` and `print("str1")` will be `str1`.
4. The output of the Python code `str1 = "Programming"` and `print(str1[3:8])` will be `gramm`.
5. The output of the Python code `str1 = "save paper,save trees"` and `str1.find("save")` will be the index of the first occurrence of "save" in the string, which is `0`.
6. To get "Aryan" as output from the string `str1 = "Vishv,Aryan,Devarsh"`, the correct code would be `print(str1[-13:-8])`.
7. The Python code `str1 = "LJ University"` and `print(len(str1))` will return `13`.
8. The output of the Python statement `""abcdef"[2:8]` will be `cdef`.
9. The output of the Python statement `print('new'   'line')` will be `newline`.
10. The output of the Python code `str1 = "hello world"` and `str1[::-1]` will be `dlrow olleh`.
11. The output of the Python code:

```
x = ['ab', 'cd']
for i in x:
    i.upper()
print(x)
```

will be `['ab', 'cd']`. However, the strings won't be converted to uppercase in the list `x` because `i.upper()` doesn't modify `i` in place; it returns a modified string that isn't stored anywhere in this loop.

1. The output of the Python code:

```
x = 'abcd'
for i in range(len(x)):
    i.upper()
print(x)
```

will be `error`. The `upper()` method is applied to the iterator variable `i` in the loop, but 'i' will be an integer and not a string.

1. The output of the Python code `print('abcd1234'.isalnum())` will be `True`.

2. The correct output of the following string operations:

```
str1 = "my isname isisis jameis isis bond"
sub = "is"
print(str1.count(sub, 5))
```

will be 6. The `count()` method counts occurrences of the substring "is" in `str1` starting from index 5.

1. The output of the string comparison will be:

```
print("John" > "Jhon")   # True
print("Emma" < "Emm")    # False
```

1. The correct output of the following string operations:

```
str1 = 'Welcome'
print(str1[:6] + 'LJIET')
```

will be `WelcomLJIET`.

1. The output of the code:

```
str1 = "LJIET"
print(str1[1:4], str1[:5], str1[4:], str1[0:-1], str1[:-1])
```

will be `JIE LJIET T LJIE LJIE`.

1. A Python tuple is represented as `(1, 2, 3)`.
2. The creation of a tuple can be done using `tuple1 = ("a", "b")`.
3. The output type of the code:

```
aTuple = ("Orange")
print(type(aTuple))
```

will be `<class 'str'>`.

256. Function with Tuple Unpacking

```
def practice(tup):
    a, b, c = tup
    return b

aTuple = "Orange", 30, "White"
print(practice(aTuple))
```

Explanation: The function `practice` takes a tuple `tup` as its argument and unpacks it into three variables `a`, `b`, and `c`. The function then returns the value of `b`. In the provided code, `aTuple` is a tuple with three elements: "Orange", 30, and "White". When `aTuple` is passed to the `practice` function, it unpacks the tuple, and the function returns the value of `b`, which is 30. Therefore, the output of the code is `30`.

258. Tuple Slicing

```
t = (1, 2, 4, 3)
print(t[1:3])
```

Explanation: The output of the code is the slice of the tuple `t` from index 1 (inclusive) to index 3 (exclusive). Therefore, it will print `(2, 4)`.

259. Tuple Slicing

```
t = (1, 2, 4, 3)
print(t[1:-1])
```

Explanation: Similar to the previous question, the output is the slice of the tuple `t` from index 1 (inclusive) to the second-to-last index (exclusive). Thus, it will print `(2, 4)`.

260. Tuple Repetition

```
t = (4, 6)
print(2 * t)
```

Explanation: The output is the result of repeating the tuple `t` two times using the multiplication operator. Therefore, it will print `(4, 6, 4, 6)`.

261. Tuple Repetition and Concatenation

```
tuple1 = (2, 3, 4)
tuple3 = tuple1 * 2
print(tuple3)
```

Explanation: The output is the result of repeating the tuple `tuple1` two times using the multiplication operator. Therefore, it will print `(2, 3, 4, 2, 3, 4)`.

262. Tuple Comparison

```
t1 = (1, 2, 4, 3)
t2 = (1, 2, 3, 4)
print(t1 < t2)
```

Explanation: The output is `False` because the tuples are compared element-wise, and the first differing element in the comparison is `(4, 3)` in `t1`, which is greater than the corresponding elements `(3, 4)` in `t2`.

263. Tuple Append (Error)

```
my_tuple = (1, 2, 3, 4)
my_tuple.append((1, 2, 3))
print(len(my_tuple))
```

Explanation: Tuples are immutable in Python, which means you cannot append elements to them. The `append()` method is not available for tuples, and attempting to use it will result in an AttributeError.

264. Data Type of (1)

```
# Assuming the code is referring to a single-element tuple
print(type((1)))
```

Explanation: The output will be `<class 'int'>`. Without a trailing comma, the parentheses are interpreted as mathematical grouping rather than as indicating a tuple.

265. Tuple Slicing

```
a = (1, 2, 3, 4)
print(a[1:-1])
```

Explanation: The output is the slice of the tuple `a` from index 1 (inclusive) to the second-to-last index (exclusive). Thus, it will print `(2, 3)`.

266. Tuple Comparison

```
a = (1, 2, (4, 5))
b = (1, 2, (3, 4))
print(a < b)
```

Explanation: The output is `False` because tuples are compared element-wise. The first elements are equal, the second elements are equal, but the third elements are compared as `(4, 5)` and `(3, 4)`, and `(4, 5)` is greater.

267. Tuple Repetition (String)

```
a = ("Check",) * 3
print(a)
```

Explanation: The output is a tuple consisting of the string "Check" repeated three times. Therefore, it will print 'CheckCheckCheck'

268. Tuple Summation

```
a = (2, 3, 4)
sum(a, 3)
```

Explanation: The `sum()` function in Python returns the sum of elements in a sequence (such as a tuple), plus an optional initial value. In this case, it computes the sum of elements in the tuple `(2, 3, 4)` and adds `3` to the result. The output will be `12` (2 + 3 + 4 + 3 = 12).

269. Deleting a Tuple (Validity)

```
a = (1, 2, 3, 4)
del a
```

Explanation: Yes, the code is valid. It deletes the tuple `a` using the `del` keyword. After executing `del a`, the tuple `a` will no longer exist in memory.

270. Slicing a Tuple

```
a = (0, 1, 2, 3, 4)
b = slice(0, 2)
a[b]
```

Explanation: The `slice()` function creates a slice object, and `a[b]` uses this slice object `b` to extract elements from the tuple `a`. It selects elements from index `0` to `2` (exclusive), so the output will be `(0, 1)`.

272. Tuple Slicing with Negative Index

```
tupl = ("annie", "hena", "sid")
print(tupl[-3:0])
```

Explanation: Slicing from negative indices doesn't directly work to select elements up to a certain index. The result would be an empty tuple because the negative index `-3` means the third element from the end, and `0` is the stop index (exclusive), resulting in an empty slice.

274. Repetition of an Empty Tuple

```
tupl = ()
tupl1 = tupl * 2
print(len(tupl1))
```

Explanation: The code creates an empty tuple `tupl` and repeats it twice (`* 2`). The output will be `0` since the resulting tuple is also empty after repetition.

275. Modifying a Tuple with a Mutable Element

```
tupl = ([2, 3], "abc", 0, 9)
tupl[0][1] = 1
print(tupl)
```

Explanation: The code attempts to modify the first element of the tuple `tupl`, which is a list `([2, 3])`. This operation is allowed because lists are mutable. It changes the second element of the list from `3` to `1`. The output will be `([2, 1], 'abc', 0, 9)`.

278. Tuple Slicing Output

```
aTuple = (10, 20, 30, 40, 50, 60, 70, 80)
print(aTuple[2:5], aTuple[:4], aTuple[3:])
```

Explanation: This code prints three slices of the tuple `aTuple`.

- `aTuple[2:5]` prints the elements from index `2` to `5` (exclusive): `(30, 40, 50)`.
- `aTuple[:4]` prints the elements up to index `4` (exclusive): `(10, 20, 30, 40)`.

- `aTuple[3:]` prints the elements from index `3` onwards: `(40, 50, 60, 70, 80)`.

279. Iterating Through a Tuple

```
t = (1, 2, 4, 3, 8, 9)
for i in range(0, len(t), 2):
    print(t[i], end=" ")
```

Explanation: This code iterates through the tuple `t` using `range()` with a step of `2`. It prints the elements at the indices `0`, `2`, and `4`. The output will be `1 4 8`.

280. Tuple Equality Comparison

```
t1 = (1, 2)
t2 = (2, 1)
x = (t1 == t2)
print(x)
```

Explanation: This code compares the equality of tuples `t1` and `t2`. As they contain different elements in different orders, the output will be `False`.

281. Count Method for a Substring in a String

```
str1 = "Hello World! Hello Hello"
str1.count("Hello", 12, 25)
```

Explanation: The `count()` method counts the occurrences of the substring "Hello" within the string `str1`, considering the substring's presence between indices `12` and `25` (inclusive). The output will be `2` as there are occurrences within that range.

282. Finding Character Indices in a String

```
a = "Hello Welcome to the Python"
print(a.find("z"))
print(a.index("z"))
```

Explanation: Both `find()` and `index()` methods are used to locate the index of a substring within a string. If the substring is not found, `find()` returns `-1`, and `index()` raises a `ValueError`. In this case, as there's no "z" in the string, both `find()` and `index()` will return `-1`, but `index()` will raise a `ValueError` as it cannot find the substring.

283. Accessing Elements in a Tuple Using Indices

```


t1 = (1, 2, 3, 4, 5, 6, 7)
print(t1[t1[1] + t1[-4]])
```

Explanation: This code accesses an element in `t1` using an index computed from elements within `t1`. `t1[1]` is `2`, and `t1[-4]` is `4`, so the final index is `2 + 4 = 6`. The output will be the element at index `6` of `t1`, which is `7`.

284. Checking Palindrome using Recursion

```
def check(s):
    if len(s) <= 1:
        return True
    else:
        return s[0] == s[-1] and check(s[1:-1])

print(check('saippuakivkauppias'))
```

Explanation: This code checks whether the input string is a palindrome using recursion. It checks if the first and last characters are the same and recursively checks the inner substring excluding the first and last characters. The input string `'saippuakivkauppias'` is not a palindrome, so the output will be `False`.

285. Empty Tuple Repetition

```
t = ()
t1 = t * 10
print(len(t1))
```

Explanation: This code creates an empty tuple `t` and repeats it ten times (`* 10`). As the original tuple is empty, the resulting tuple `t1` after repetition will also be empty, so the output will be `0`.

286

```
def tup(T):
    print(T[T.index(5)], end = "")
    print(T[T[T[6]-3]-6])

T = (1, 2, 3, 4, 5, 6, 7, 8)
tup(T)
```

Explanation:

- `T.index(5)` returns the index of the first occurrence of `5` in the tuple `T`, which is `4`.
- `T[6]` is `7`.
- `T[7-3]` is `T[4]`, which is `5`.
- `T[5-6]` is `T[-1]`, which is `8`.

The output will be `5 8`.

287

```
s = "Th*is is$ nothi&&ng b#ut excerc(is)e"
change = str.maketrans("(),@$%^&*_-", "")
s.translate(change)
print(s)
```

Explanation: The `str.translate()` method returns a translated string, but in this code, the result of the translation is not assigned back to the variable `s`. So, the original string `s` is printed without any modifications.

The output will be the original string `"Th*is is$ nothi&&ng b#ut excerc(is)e"`.

289

```
s = "ball"
r = ""
for i in s:
    r = i.upper() + r
print(r)
```

Explanation: This code iterates through the string `s` and concatenates the uppercase version of each character to the beginning of string `r`. Therefore, the output will be the reversed string of `s` in uppercase, which is `"LLAB"`.

290

```
s = 'I love my INDIA'
print(s[-1] + s[3:4] + s[7:9] + s[-3:-1] + s[-1:-3:-1] + s[5:9] +
s[10:])
```

Explanation:

- `s[-1]` returns the last character which is `"A"`.
- `s[3:4]` returns `"o"`.
- `s[7:9]` returns `"my"`.
- `s[-3:-1]` returns `"DI"`.
- `s[-1:-3:-1]` returns the reverse of the last two characters which is `"AI"`.
- `s[5:9]` returns `"e my"`.
- `s[10:]` returns `"INDIA"`.

Concatenating these parts together results in `"AomyDIAIe myINDIA"`.

291

```
m = (1, [1, 2], 3, 4)
m[1][1] = 5
type(m)
```

Explanation: This code creates a tuple `m` with nested lists. It then modifies the element at index `1` of the tuple (which is a list `[1, 2]`) by changing its second element to `5`. Finally, it retrieves the type of `m`, which will be `<class 'tuple'>`.

292

When using `find()`, if the substring is not present in the string, `-1` is returned.

293

```python
s = "blog"
for i in range(-1, -len(s), -1):
    print(s[i], end="$")
```

Explanation: This code iterates backward through the string `s` and prints each character followed by `$`. It starts from the last character and prints each character in reverse order followed by `$`. The output will be `g$o$l$`.

294

```python
print("A#B#C#D#E".split("#", 2))
```

Explanation: The `split()` method divides the string based on the delimiter `"#"`. The `2` as the second argument specifies that the splitting should be done only twice. Therefore, the output will be `['A', 'B', 'C#D#E']`.

295

```python
for i in range(len("python"), 12, 2):
    print("python"[i - 6], end="")
```

Explanation: The `range()` starts from the length of the string `"python"` which is `6`. It iterates from `6` to `12` with a step of `2`. Therefore, it prints characters from the string `"python"` indexed by `0`, `2`, and `4`, which are `pto`.

296

```python
x = 'abcd'
for i in x:
    i.isupper()
print(x)
```

Explanation: This code iterates through each character in the string `x` but does not capture or utilize the result of the `i.isupper()` method. It doesn't change the original string `x`, so it prints `"abcd"`.

297

```python
t = (1, 2, 4, 3, 6, 8, 4)
t[1:-1:-1]
```

Explanation: The slicing `t[1:-1:-1]` with a negative step `-1` tries to get a slice starting from index `1` and ending at index `-1` (exclusive) in reverse order. However, since the start index is greater than the end index, it doesn’t retrieve any elements. The output will be an empty tuple `()`.

298

```
my_tuple = (1, 2, 3, 4)
my_tuple.append((1, 2, 3))
print(len(my_tuple))
```

Explanation: Tuples are immutable in Python and do not have an `append()` method. The code will raise an `AttributeError` because tuples do not support the `append()` operation.

299

```
def enc(st):
    encoded = ""
    c = 1
    ld = st[0]
    for i in range(1, len(st)):
        if ld == st[i

]:
            c += 1
        else:
            encoded += str(c) + ld
            c = 1
            ld = st[i]
    encoded += str(c) + ld
    return encoded

st = "AAABBACCAA"
print(enc(st))
```

Explanation: This code performs run-length encoding on the input string `st`. It counts the consecutive characters and represents them as a single character followed by its count. The output will be `3A2B1A2C2A`.

300

```
s = "aa"
s.strip("a")
print(s)
```

Explanation: The `strip()` method returns a new string after removing leading and trailing characters. However, since `strip()` doesn't modify the string in place, and there are no leading or trailing characters in `"aa"` other than `"a"`, the output will be `"aa"`.

301

```
s = "1234ABCvhghbbv"
v = s.maketrans("abc", "vvv")
print(s.translate(v))
```

Explanation: The `maketrans()` function creates a translation table that maps `a`, `b`, and `c` to `v`, `v`, and `v` respectively. Then, the `translate()` method applies this translation to the string `s`. Therefore, the output will be `"1234ABCvvhghvvv"`.

302

```
shift = 1
n = 12345
s = str(n)
x = s[shift:] + s[:shift]
print(x)
```

Explanation: This code performs a circular shift on the string representation of the number `n`. It moves `shift` positions from the start to the end of the string and concatenates it with the characters from the start to `shift`. For `shift = 1`, the output will be `"23451"`.

303

```
str1 = "LJ'University"
print(len(str1))
```

Explanation: The `len()` function counts the number of characters in the string `str1`. The output will be `13`.

304

```
tupl = ()
tupl1 = tupl * 2
print(len(tupl1))
```

Explanation: The tuple `tupl` is empty, and when multiplied by `2`, it results in an empty tuple again. Therefore, the output will be `0`.

305

```
A = (5, 3, 2)
B = (5, 3, 2)
print(len(A + B * 3))
```

Explanation:

- `B * 3` multiplies the tuple `(5, 3, 2)` by `3`, resulting in `(5, 3, 2, 5, 3, 2, 5, 3, 2)`.
- `A + B` concatenates the tuples `(5, 3, 2)` and `(5, 3, 2, 5, 3, 2, 5, 3, 2)`, resulting in `(5, 3, 2, 5, 3, 2, 5, 3, 2, 5, 3, 2)`.
- `A + B * 3` concatenates the two tuples, resulting in `(5, 3, 2, 5, 3, 2, 5, 3, 2, 5, 3, 2)`.

The length of this tuple will be `12`.

Unit - 5 MCQs

333. What is the output when we execute `list("hello")`?

```
print(list("hello"))
```

Output:

```
['h', 'e', 'l', 'l', 'o']
```

Explanation: `list("hello")` converts the string `"hello"` into a list of individual characters.

334. Suppose `listExample` is `['h', 'e', 'l', 'l', 'o']`, what is `len(listExample)`?

```
listExample = ['h', 'e', 'l', 'l', 'o']
print(len(listExample))
```

Output:

```
5
```

Explanation: `len()` returns the number of elements in the list.

335. Suppose `list1` is `[1, 3, 2]`, What is `list1 * 2`?

```
list1 = [1, 3, 2]
print(list1 * 2)
```

Output:

```
[1, 3, 2, 1, 3, 2]
```

Explanation: The `*` operator with a list duplicates its elements.

336. Output of the following Python code:

```
names1 = ['Amir', 'Bala', 'Chales']
if 'amir' in names1:
    print(1)
else:
    print(2)
```

Output:

```
2
```

Explanation: The comparison is case-sensitive. `'amir'` is not in the list because of the lowercase 'a'.

337. Output of the following Python code:

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
print(len(list1 + list2))
```

Output:

```
8
```

Explanation: `len(list1 + list2)` returns the length of the concatenated list.

339. Suppose `list1` is `[4, 2, 2, 4, 5, 2, 1, 0]`, Which of the following is correct syntax for slicing operation?

Correct syntax for slicing operations is `list[start:stop]`.

340. Suppose `list1` is `[2, 33, 222, 14, 25]`, What is `list1[-1]`?

```
list1 = [2, 33, 222, 14, 25]
print(list1[-1])
```

Output:

```
25
```

Explanation: `list1[-1]` accesses the last element of the list.

341. Suppose `list1` is `[2, 33, 222, 14, 25]`, What is `list1[:-1]`?

```
list1 = [2, 33, 222, 14, 25]
print(list1[:-1])
```

Output:

```
[2, 33, 222, 14]
```

Explanation: `list1[:-1]` returns all elements except the last one.

342. Output of the following Python code:

```
names = ['Amir', 'Bear', 'Charlton', 'Daman']
print(names[-1][-1])
```

Output:

```
'n'
```

Explanation: `names[-1]` gets the last element, which is `'Daman'`, and `names[-1][-1]` accesses the last character of that string.

344. Output of the following Python code:

```
names1 = ['Amir', 'Bear', 'Charlton', 'Daman']
names2 = names1
names3 = names1[:]
names2[0] = 'Alice'
names3[1] = 'Bob'
sum = 0
for ls in (names1, names2, names3):
    if ls[0] == 'Alice':
        sum += 1
    if ls[1] == 'Bob':
        sum += 10
print(sum)
```

Output:

```
12
```

Explanation: `names2` and `names3` are copies of `names1`. Modifying `names2` and `names3` will affect `names1` as well.

345. Output of the following Python code:

```
names1 = ['Amir', 'Bear', 'Charlton', 'Daman']
names2 = names1
names3 = names1
names2[0] = 'Alice'
names3[1] = 'Bob'
sum = 0
for ls in (names1, names2, names3):
    if ls[0] == 'Alice':
        sum += 1
    if ls[1] == 'Bob':
        sum += 10
print(sum)
```

Output:

```
33
```

Explanation: In this, modifying `names2` and `names3` affects `names1`, `names2` and `names3` due to reference assignment.

346. Output of the following Python code:

```
lst = [3, 4, 6, 1, 2]
lst[1:2] = [7, 8]
print(lst)
```

Output:

```
[3, 7, 8, 6, 1, 2]
```

Explanation: Slicing replaces the elements from index `1` to `2` (exclusive) with `[7, 8]`.

347. Output of the below Python code:

```
list1 = [8, 0, 9, 5]
print(list1[::-1])
```

Output:

```
[5, 9, 0, 8]
```

Explanation: `list1[::-1]` reverses the order of the list elements.

348. Suppose `list1 = [0.5 * x for x in range(0, 4)]`, what is `list1`?

```
list1 = [0.5 * x for x in range(0, 4)]
print(list1)
```

Output:

```
[0.0, 0.5, 1.0, 1.5]
```

Explanation: `list1` is a list comprehension that generates elements by multiplying each number in the range `[0, 1, 2, 3]` by `0.5`.

349. To add a new element to a list we use which command?

The command used to add a new element to a list is `append()`. For instance:

```
my_list = [1, 2, 3]
my_list.append(4)  # Adds 4 to the end of the list
print(my_list)  # Output: [1, 2, 3, 4]
```

350. Output of the following Python code:

```
numbers = [1, 2, 3, 4]
numbers.append([5, 6, 7, 8])
print(len(numbers))
```

Output:

```
5
```

Explanation: `append()` adds the list `[5, 6, 7, 8]` as a single element to `numbers`, so the length of `numbers` becomes 5.

351. Output of the following Python code:

```
a = [1, 2, 3]
b = a.append(4)
print(a)
print(b)
```

Output:

```
[1, 2, 3, 4]
None
```

Explanation: `a.append(4)` modifies list `a` in place and returns `None`. Hence, `b` holds the value `None`.

352. What is returned by the following function?

```
def list_transformation():
    alist = [4, 2, 8, 6, 5]
    blist = []
    for item in alist:
        blist.append(item + 5)
    return blist
```

This function returns a new list `blist` containing elements obtained by adding 5 to each element in `alist`.

353. Output of the following code:

```
def mystery(num_list):
    out = []
    for num in num_list:
        if num > 10:
            out.append(num)
    return out
print(mystery([5, 10, 15, 20]))
```

Output:

```
[15, 20]
```

Explanation: The function `mystery` filters elements greater than 10 from the given list.

354. Suppose `list1` is `[3, 4, 5, 20, 5, 25, 1, 3]`, what is `list1.count(5)`?

```
list1 = [3, 4, 5, 20, 5, 25, 1, 3]
print(list1.count(5))
```

Output:

```
2
```

Explanation: `list1.count(5)` returns the number of occurrences of `5` in `list1`, which is `2`.

355. Output of the following Python code:

```
x = [1, 2, 3]
y = [7, 8, 9]
x.extend(y)
print(x)
```

Output:

```
[1, 2, 3, 7, 8, 9]
```

Explanation: `extend()` method appends the elements of list `y` to the end of list `x`.

356. Output of the following Python code:

```
x = [1, 2, 3]
y = "789"
x.extend(y)
print(x)
```

Output:

```
[1, 2, 3, '7', '8', '9']
```

Explanation: When extending a list with a string using `extend()`, each character of the string becomes a separate element in the list.

357. What will be the value of 'result' in the following Python program?

```
list1 = [1, 2, 3, 4]
list2 = [2, 4, 5, 6]
list3 = [2, 6, 7, 8]
result = list()
result.extend(i for i in list1 if i not in (list2 + list3) and i not
in result)
result.extend(i for i in list2 if i not in (list1 + list3) and i not
in result)
result.extend(i for i in list3 if i not in (list1 + list2) and i not
in result)
print(result)
```

Output:

```
[1, 3, 5, 7, 8]
```

Explanation: `result` contains elements that are present in only one list among `list1`, `list2`, and `list3`.

358. Suppose `list1` is `[3, 4, 5, 20, 5]`, what is `list1.index(5)`?

```
list1 = [3, 4, 5, 20, 5]
print(list1.index(5))
```

Output:

```
2
```

Explanation: `list1.index(5)` returns the index of the first occurrence of `5` in `list1`, which is at index `2`.

359. To insert `5` to the third position in `list1`, we use which command?

To insert an element at a specific index in a list, we use the `insert()` method:

```
list1 = [3, 4, 5, 20, 5]
list1.insert(2, 5)
print(list1)
```

Output:

```
[3, 4, 5, 5, 20, 5]
```

Explanation: `list1.insert(2, 5)` inserts `5` at index `2` in `list1`.

360. Output of the following Python code:

```
veggies = ['carrot', 'broccoli', 'potato', 'asparagus']
veggies.insert(veggies.index('broccoli'), 'celery')
print(veggies)
```

Output:

```
['carrot', 'celery', 'broccoli', 'potato', 'asparagus']
```

Explanation: `veggies.index('broccoli')` gives the index of `'broccoli'`, and `veggies.insert(index, 'celery')` inserts `'celery'` before `'broccoli'` in `veggies`.

361. What will be the result after the execution of the following Python code?

```
list1 = [3, 2, 5, 7, 3, 6]
list1.pop(3)
print(list1)
```

Output:

```
[3, 2, 5, 3, 6]
```

Explanation: `list1.pop(3)` removes the element at index `3` from `list1`.

362. To remove the string “hello” from `list1`, we use which command?

To remove an element from a list, you can use the `remove()` method:

```
list1 = ["hi", "hello", "bye"]
list1.remove("hello")
print(list1)
```

Output:

```
['hi', 'bye']
```

Explanation: `list1.remove("hello")` removes the specific element `"hello"` from `list1`.

363. Suppose `list1` is `[3, 4, 5, 20, 5, 25, 1, 3]`, what is `list1` after `list1.reverse()`?

```
list1 = [3, 4, 5, 20, 5, 25, 1, 3]
list1.reverse()
print(list1)
```

Output:

```
[3, 1, 25, 5, 20, 5, 4, 3]
```

Explanation: `list1.reverse()` reverses the elements in `list1` in-place.

364. Output of the below Python code:

```
numbers = [1, 3, 4, 2]
numbers.sort()
print(numbers)
```

Output:

```
[1, 2, 3, 4]
```

Explanation: `numbers.sort()` sorts the list `numbers` in ascending order.

365. Output of the below Python code:

```
decimalnumber = [2.01, 2.00, 3.67, 3.28, 1.68]
decimalnumber.sort()
print(decimalnumber)
```

Output:

```
[1.68, 2.0, 2.01, 3.28, 3.67]
```

Explanation: `decimalnumber.sort()` sorts the list `decimalnumber` in ascending order.

366. Output of the below Python code:

```
words = ["Geeks", "For", "Geeks"]
words.sort()
print(words)
```

Output:

```
['For', 'Geeks', 'Geeks']
```

Explanation: `words.sort()` sorts the list alphabetically.

367. Output of the following Python code snippet:

```
d = {"john": 40, "peter": 45}
"john" in d
```

Output:

```
True
```

Explanation: `"john"` is a key in the dictionary `d`.

368. Output of the following Python code snippet:

```
d1 = {"john": 40, "peter": 45}
d2 = {"john": 466, "peter": 45}
d1 == d2
```

Output:

```
False
```

Explanation: `d1` and `d2` have different values associated with the key `"john"`.

369. Output of the following Python code snippet:

```
d = {"john": 40, "peter": 45}
d["john"]
```

Output:

```
40
```

Explanation: Accessing the value associated with the key `"john"` in dictionary `d`.

370. Output of the following Python code:

```
d = {0: 'a', 1: 'b', 2: 'c'}
for i in d:
    print(i)
```

Output:

```
0
1
2
```

Explanation: When iterating over a dictionary, it iterates over its keys by default.

371. To obtain the number of entries in dictionary `d`, which command do we use?

To obtain the number of entries in a dictionary, you use the `len()` function:

```
d = {"john": 40, "peter": 45}
entries = len(d)
print(entries)  # This will output the number of entries in the
dictionary, which is 2 in this case.
```

372. Output of the following Python code snippet:

```
d = {"john": 40, "peter": 45}
print(list(d.keys()))
```

Output:

```
['john', 'peter']
```

Explanation: `d.keys()` returns a view object of keys in the dictionary `d`, which is then converted to a list.

373. Which of the following is not a declaration of the dictionary?

The correct answer is the following statement:

```
{1,”A”,2,”B”}
```

This creates a set, not a dictionary.

374. Output of the following Python code:

```
a = dict()
a[1]
```

Output:

```
KeyError: 1
```

Explanation: Trying to access a non-existent key (`1`) in an empty dictionary raises a `KeyError`.

375. Output of the following Python code:

```
a = {}
a[2] = 1
a[1] = [2, 3, 4]
print(a[1][1])
```

Output:

```
3
```

Explanation: `a[1]` is a list `[2, 3, 4]`, and `a[1][1]` accesses the element at index `1` within that list, which is `3`.

376. Output of the following Python code snippet:

```
numbers = {}
letters = {}
comb = {}
numbers[1] = 56
numbers[3] = 7
letters[4] = 'B'
comb['Numbers'] = numbers
comb['Letters'] = letters
print(comb)
```

Output:

```
{'Numbers': {1: 56, 3: 7}, 'Letters': {4: 'B'}}
```

Explanation: Three dictionaries (`numbers`, `letters`, and `comb`) are defined and then combined into the `comb` dictionary as values with keys `'Numbers'` and `'Letters'`.

377. Which of the following statements create a dictionary?

The correct statement that creates a dictionary is:

```
{'key': 'value'}
```

This syntax represents a dictionary with a single key-value pair.

378. What will be the output of the following Python code?

```
a = {1: "A", 2: "B", 3: "C"}
for i in a:
    print(i, end=" ")
```

- **Answer:** This will output `1 2 3`.

379. What will be the output of the following Python code?

```
text = {1: "geeks", 2: "for"}
text.clear()
print(text)
```

- **Answer:** This will output an empty dictionary: `{}`.

380. What will be the output of the following Python code?

```
a = {1: "A", 2: "B", 3: "C"}
b = a.copy()
b[2] = "D"
print(a)
```

- **Answer:** The output will be `{1: 'A', 2: 'B', 3: 'C'}`. The `copy()` method creates a shallow copy of the dictionary.

381. What is the output of the following piece of code?

```
a = {1: "A", 2: "B", 3: "C"}
print(a.get(1, 4))
```

- **Answer:** The output will be `'A'`. The `get()` method returns the value associated with the specified key.

382. What is the output of the following piece of code?

```
a = {1: "A", 2: "B", 3: "C"}
print(a.get(5, 4))
```

- **Answer:** The output will be `4`. Since the key `5` does not exist in the dictionary, the `get()` method returns the default value `4`.

383. What will be the output of the following Python code snippet?

```
a = {1: "A", 2: "B", 3: "C"}
for i, j in a.items():
    print(i, j, end=" ")
```

- **Answer:** This will output `1 A 2 B 3 C`.

384. What will be the output of the following Python code?

```
a = {1: "A", 2: "B", 3: "C"}
print(a.items())
```

- **Answer:** The output will be `dict_items([(1, 'A'), (2, 'B'), (3, 'C')])`.

385. What will be the output of the following Python code?

```
Dictionary1 = {'A': 'Geeks', 'B': 'For', 'C': 'Geeks'}
print(Dictionary1.keys())
```

- **Answer:** The output will be `dict_keys(['A', 'B', 'C'])`.

386. What will be the output of the following Python code?

```
a = {1: "A", 2: "B", 3: "C"}
b = {4: "D", 5: "E"}
a.update(b)
print(a)
```

- **Answer:** The output will be `{1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E'}`. The `update()` method adds elements from another dictionary into the current dictionary.

387. What will be the output of the following Python code?

```
dictionary = {"raj": 2, "striver": 3, "vikram": 4}
print(dictionary.values())
```

- **Answer:** The output will be `dict_values([2, 3, 4])`.

389. Which of the following statements is used to create an empty set?

- **Answer:** `set()` is used to create an empty set.

390. What will be the output of the following Python code?

```
s = {5, 6}
s * 3
```

- **Answer:** This code will raise an error. Multiplication (`*`) operation is not supported for sets.

391. What will be the output of the following Python code?

```
a = {5, 6, 7, 8}
b = {7, 5, 6, 8}
a == b
```

- **Answer:** The output will be `True`. Both sets contain the same elements, so they are equal irrespective of the order.

392. What will be the output of the following Python code?

```
a = {4, 5, 6}
b = {2, 8, 6}
a + b
```

- **Answer:** This code will raise an error. The `+` operator is not supported for sets.

393. What will be the output of the following Python code?

```
a = {4, 5, 6}
b = {2, 8, 6}
a - b
```

- **Answer:** The output will be `{4, 5}`. The `-` operator returns the difference of sets.

395. What will be the output of the following Python code, if `s1 = {1, 2, 3}`?

```
s1.issubset(s1)
```

- **Answer:** This will output `True`. `s1` is a subset of itself, as every set is considered a subset of itself.

396. If we have two sets, `s1` and `s2`, and we want to check if all the elements of `s1` are present in `s2` or not, we can use the function:

- **Answer:** To check if all elements of `s1` are present in `s2`, we can use the function `s2.issuperset(s1)`.

397. What will be the output of the following Python code?

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.union(y)
print(z)
```

- **Answer:** The output will be `{'apple', 'cherry', 'banana', 'google', 'microsoft'}`. The `union()` method combines elements of both sets removing duplicates.

398. What will be the output of the following Python code?

```
s1 = {1, 2, 3}
s2 = {2, 3}
print(s1.intersection(s2))
```

- **Answer:** The output will be `{2, 3}`. The `intersection()` method returns the common elements between `s1` and `s2`.

399. What will be the output of the following Python code?

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.difference(y)
print(z)
```

- **Answer:** The output will be `{'banana', 'cherry'}`. The `difference()` method returns elements that are in `x` but not in `y`.

400. What will be the output of the following Python code?

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.symmetric_difference(y)
print(z)
```

- **Answer:** The output will be `{'banana', 'cherry', 'google', 'microsoft'}`. The `symmetric_difference()` method returns elements that are in either `x` or `y` but not in both.

401. What will be the output of the following Python code?

```
fruits = {"apple", "banana", "cherry"}
x = fruits.copy()
print(x)
```

- **Answer:** The output will be `{'apple', 'banana', 'cherry'}`. The `copy()` method creates a shallow copy of the set.

402. What will be the output of the following Python code?

```python
y = 6
z = lambda x: x * y
print(z(8))
```

- **Answer:** The output will be `48`. The lambda function `z` multiplies the input `x` with the value of `y`.

403. What will be the output of the following Python code?

```python
lamb = lambda x: x ** 3
print(lamb(5))
```

- **Answer:** The output will be `125`. The lambda function `lamb` cubes the input `x`.

404. What will be the output of the following Python code?

```python
def writer():
    title = 'Sir'
    name = (lambda x: title + ' ' + x)
    return name
who = writer()
print(who('Arthur'))
```

- **Answer:** The output will be `Sir Arthur`. The lambda function `name` concatenates `'Sir'` with the given input.

405. What will be the output of the following Python code?

```python
min = (lambda x, y: x if x < y else y)
print(min(101*99, 102*98))
```

- **Answer:** The output will be `9996`. The lambda function `min` finds the minimum value between `101*99` and `102*98`.

406. What will be the output of the following Python code?

```python
def current_date(**kwargs):
    for i in kwargs:
        print(i)

current_date(date=2-1-2023)
```

The `current_date` function is defined to accept keyword arguments using `**kwargs`. Inside the function, it iterates through the keys of the provided keyword arguments (`kwargs`) and prints them.

However, in the line where the function is called (`current_date(date=2-1-2023)`), the expression `2-1-2023` performs arithmetic operations (`2-1` equals `1`, and then `1-2023` equals `-`

2022). This numeric value -2022 is then passed as an argument named date to the current_date function.

Consequently, when this function is executed, it will print the string representation of the argument passed ('date'). Therefore, the output of this code will be:

```
date
```

407. What will be the output of the following Python code?

```
def f1(*m):
    sum1 = len(m)
    for i in m:
        sum1 += i
    return sum1
x = f1(1, 2, 3, (4,)==(4,), (5,)==(5))
print(x)
```

- **Answer:** m will store (1, 2, 3, True, False) and will give answer 12.

408. What will be the output of the following Python code?

```
def F(B, A=3, *C, **D):
    sum_ = A + B
    for i in C:
        sum_ += i
    for i in D.values():
        sum_ += i
    return sum_
print(F(1, 5, 7, 4, 3, e=1, f=2))
```

- **Answer:** The output will be 23. The function adds the values passed and assigns default values if not given.

411.

```
numberGames = {}
numberGames[(1,2,4)] = 8
numberGames[(4,2,1)] = 10
numberGames[(1,2)] = 12
sum = 0
for k in numberGames:
    sum += numberGames[k]
print (len(numberGames) + sum)
```

Explanation:

- numberGames is a dictionary with three keys: (1,2,4), (4,2,1), and (1,2), each assigned a value.
- The loop calculates the sum of all values in numberGames.

- `len(numberGames)` returns the number of items in the dictionary.
- Finally, it prints the sum of the length of `numberGames` and the sum of its values and len of numberGames.

Output: 33

412.

```
L=[['Physics',101],['Chemistry',202],['Maths',303],45,6,'j']
print(len(L))
```

Explanation:

- `L` is a list containing various data types (lists, integers, and strings).
- `len(L)` returns the number of elements in the list.

Output: 6

413.

```
set1={2,3}
set2={3,2}
set3={2,1}

if(set1==set2):
    print("yes")
else:
    print("no")

if(set1==set3):
    print("yes")
else:
    print("no")
```

Explanation:

- `set1`, `set2`, and `set3` are sets with different element orders but the same elements.
- Both `set1` and `set2` contain the same elements, so the first condition prints "yes".
- `set1` and `set3` don't have the same elements, so the second condition prints "no".

Output:

```
yes
no
```

414.

```
D={1:"Amit",2:"Suman",3:"Ravi",4:"Anuj"}
print(max(D.values()))
```

Explanation:

- `D` is a dictionary with integer keys and string values.
- `D.values()` returns a collection of values.
- `max()` finds the maximum value among these strings.

Output: Suman

415.

```
L = ['Arnold', 'Bootboggler', 'Christi', 'Dickinson']
print(L[-1][-1])
```

Explanation:

- `L[-1]` refers to the last element of the list, which is the string `'Dickinson'`.
- `L[-1][-1]` accesses the last character of the string `'Dickinson'`.

Output: n

416.

```
D = {1: 1, 2: '2', '1': 2, '2': 3}
D['1'] = 2
print(D[D[D[str(D[1])]]])
```

Explanation:

- `str(D[1])` retrieves the value for key `1` in dictionary `D` and converts it to a string.
- `D[str(D[1])]` retrieves the value associated with the string representation of the value corresponding to key `1`.
- `D[D[str(D[1])]]` again retrieves a value from dictionary `D` using the value retrieved in the previous step as a key.
- The final `print` statement displays the value associated with the final key retrieved.

Output: 3

417.

```
L1 = []
L1.append([1, [2, 3], 4])
L1.extend([7, 8, 9])
print(L1[0][1][1] + L1[2])
```

Explanation:

- `L1` is a list.
- `L1[0][1][1]` accesses the second element of the list `[2, 3]` within the first element `[1, [2, 3], 4]` of `L1`.

- L1[2] retrieves the third element from L1.
- The code adds these two values together and prints the result.

Output: 11

418.

```
L1 = [1, 1.33, 'LJU', 0, 'N', True, 'Y', 1]
val1 = 0
val2 = ""
for x in L1:
    if(type(x) == int or type(x) == float):
        val1 += x
    elif(type(x) == str):
        val2 += x
    else:
        break
        continue
print(val1, val2)
```

Explanation:

- The loop iterates through elements in L1.
- It sums up the integer and float values in val1.
- It concatenates string values in val2.
- The loop breaks when it encounters a non-integer, non-float element.

Output: 2.33 LJUN

Let's continue with the explanations for the remaining questions:

419.

```
def sum_list(l):
    sum = 0
    for i in range(len(l)):
        if l[i] == 13 or l[i - 1] == 13:
            continue
        else:
            sum += l[i]
    return sum

l = [1, 2, 13, 2, 9, 13]
print(sum_list(l))
```

Explanation:

- sum_list is a function that iterates through the list l.
- It checks if the current element or the element before it is equal to 13. If so, it continues to the next iteration.

- Otherwise, it adds the current element to the sum.
- The function returns the sum of elements that are not 13 or are not followed by 13, so even first element of list won't be added in sum as -1 index of list is 13.

Output: `11`

420.

```
L1 = [1, 1, 2, 4, 5, 6, 2, 3, 1, 3, 5]
L2 = [8, 2, 1, 3, 8, 3, 7, 2, 0]
L = L1 + L2
S = list(set(list(L)))
S.sort()
S.reverse()
S.sort()
L.reverse()
print(S)
```

Explanation:

- L1 and L2 are lists.
- L is formed by concatenating L1 and L2.
- S is created by first converting L into a set to remove duplicates, then sorting it in descending order twice.
- Finally, L is reversed.
- The code prints the resulting sorted, reversed, and sorted set.

Output: `[0, 1, 2, 3, 4, 5, 6, 7, 8]`

421.

```
tuple = {}
tuple[(1, 2, 4)] = 8
tuple[(4, 2, 1)] = 10
tuple[(1, 2)] = 12
_sum = 0
for k in tuple:
    _sum += tuple[k]
print(len(tuple) + _sum)
```

Explanation:

- `tuple` is a dictionary with tuples as keys and integer values.
- The loop calculates the sum of all values in `tuple`.
- Finally, it prints the sum of the length of `tuple` and the sum of its values.

Output: `3 + (8 + 10 + 12) = 33`

422.

```
L = list('123456')
L[0] = L[5] = 0
L[3] = L[-2]
L[5] = 1
L[-2] = 4
L[2] = L[-1]
L[4] = L[3]
L[-1] = L[3]
print(L)
```

Explanation:

- `L` is a list created from the string `'123456'`.
- The code performs a series of assignments and modifications on `L` based on index positions.
- The print statement displays the final modified list `L`.

Output: `[0, '2', 1, '5', '5', '5']`

423.

```
l1 = ['A', 'B', 'C', 'D', 'E']
l2 = l1.copy()
l3 = l1[::-1]
l2[4] = 'G'
l3[3] = 'H'
l1[4] = l2[4]
l1[3] = l3[3]
sum = 0
for i in (l1, l2, l3):
    if i[4] == 'G':
        sum += 7
    if i[3] == 'H':
        sum += 22
    if i[2] == 'C':
        sum += 30
print(sum)
```

Explanation:

- `l1`, `l2`, and `l3` are lists with different modifications.
- The loop checks conditions based on the elements at specific indices and increments the sum accordingly.

Output: `148`

424.

```
dict = {1: 2, 3: 4, 4: 11, 5: 61, 7: 81}
print(dict[dict[3]])
```

Explanation:

- The inner `dict[3]` retrieves the value associated with the key `3` in the dictionary `dict`, which is `4`.
- The outer `dict[4]` retrieves the value associated with the key `4` in the dictionary `dict`, which is `11`.

Output: `11`

425.

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
print(len(list1 + list2 - list1 + list2))
```

Explanation:

- In this code, the attempt to subtract a list from another list (`list1 - list1`) will raise a `TypeError` since subtraction between lists isn't supported.

Error: `TypeError`

426.

```
def writer():
    title = 'Sir'
    name = (lambda x: title + ' ' * 2x)
    return name

who = writer()
print(who('Arthur'))
```

Output: `'Error'` - Error in 2x.

427.

```
from functools import *
Series = lambda n: reduce(lambda x, _: x + [x[-1] + x[-2]], range(n -
2), [0, 1])
```

Explanation:

- The lambda function `Series` uses `reduce` to generate a series of numbers.

- It starts with `[0, 1]` as the initial list and then appends the sum of the last two elements `x[-1] + x[-2]` for the specified range `range(n - 2)`.

The function generates a Fibonacci-like series based on the range provided.

428.

```
names1 = ['A', 'B', 'C', 'D']
names2 = names1
names3 = names1[:]
names2[0] = 'Aa'
names3[1] = 'BB'
sum = 0
for s in (names1, names2, names3):
    if s[0] == 'Aa':
        sum += 2
    if s[1] == 'BB':
        sum += 20
print(sum)
```

Explanation:

- `names1`, `names2`, and `names3` are lists where `names2` refers to the same list as `names1` and `names3` is a copy of `names1`.
- The loop increments the sum based on conditions related to the elements at indices `0` and `1` in these lists.

Output: `24`

429.

```
l = [[["hello", "0hel"], "bh"], "nm"]
print(l[0])
```

Explanation:

- `l` is a nested list.
- `l[0]` accesses the first element of the outermost list.

Output: `[["hello", "0hel"], "bh"]`

I'll continue with the remaining questions in the next response!

Continuing with the explanations:

430.

```
nums = [3, 5, 16, 27]
some_nums = list(filter(lambda num: 5 <= num < 27, nums))
print(some_nums)
```

Explanation:

- `filter` applies the lambda function to each element in `nums`.
- The lambda function filters elements that satisfy the condition `5 <= num < 27`.

Output: `[5, 16]`

431.

```
x = {1, 2, 3, 4, 5}
y = {3, 4, 5, 6, 7}
z = {1, 3, 5, 7, 9}
print((x | y) & (x | z))
```

Explanation:

- `x | y` performs a union operation between sets `x` and `y`.
- `x | z` performs a union operation between sets `x` and `z`.
- The code then performs an intersection between the results of these union operations.

Output: `{1, 2, 3, 4, 5, 7}`

432.

```
l = [1, 2, 3, 5, 7, 8, 9, 10]
m = max(l)
print(l.index(m))
```

Explanation:

- `max(l)` finds the maximum value in the list `l`.
- `l.index(m)` returns the index of the maximum value in the list `l`.

Output: `7`

433.

```
x = ['ab', 'cd']
for i in x:
    i.upper()
print(x)
```

Explanation:

- The loop iterates through each string in the list `x`.
- `i.upper()` converts each string to uppercase, but it doesn't modify the original strings in the list. Strings are immutable in Python.

Output: `['ab', 'cd']`

434.

```
def f(l):
    l.append([1, 2, 3])
    return

l = [1, 2, 3]
print(l, end="")
f(l)
print(l)
```

Explanation:

- The function `f` appends the list `[1, 2, 3]` to the input list `l`.
- Initially, `l` is `[1, 2, 3]`.
- After calling `f(l)`, the list `l` will have an additional element appended to it.

Output: `[1,2,3] [1,2,3,[1,2,3]]`

435.

```
d = {1: "welcome", [1]: {1: 2}}
print(d[[1]])
```

Explanation:

- Dictionary keys must be immutable. Using a list (`[1]`) as a key causes a `TypeError` because lists are mutable and cannot be used as dictionary keys.

Error: `TypeError`

436.

```
l = [1, 2, [[1, 2, [1, 2], 1, 2]]]
print(l[2][0])
```

Explanation:

- `l` is a list containing another list `[1, 2, [[1, 2, [1, 2], 1, 2]]]`.
- `l[2]` accesses the third element of the outer list, which is `[[1, 2, [1, 2], 1, 2]]`.
- `l[2][0]` retrieves the first element of this inner list.

Output: `[1, 2, [1, 2], 1, 2]`

437.

```
l = [1, "m", ["a", {1: [1, 2, 3]}]]
t = {(1, 2, 3): (5)}
s = (l[2][1][1], t[(1, 2, 3)])
print(s)
```

Explanation:

- `l[2][1][1]` accesses the second element (`{1: [1, 2, 3]}`) within the nested list and then accesses the value associated with the key `1` in the dictionary.
- `t[(1, 2, 3)]` retrieves the value associated with the key `(1, 2, 3)` in the dictionary `t`.
- The values obtained from these accesses are put into a tuple `s`.

Output: `([1, 2, 3], 5)`

438.

```
l = [1, 2, (5)]
l[2] = 7
print(l)
```

Explanation:

- `l` is a list containing integers and a tuple `(5)`.
- `l[2] = 7` modifies the third element of the list, replacing the tuple `(5)` with the integer `7`.

Output: `[1, 2, 7]`

439.

```
def f(*l):
    for i in l[0]:
        sum = 0
        sum += i
    print(sum)

f([1, 2, 3])
```

Explanation:

- The function `f` receives a variable number of arguments but considers the first argument as a list.
- Inside the function, a loop iterates through the elements of the first argument list, but it incorrectly resets `sum` to zero in every iteration.
- Therefore, it prints the last element of the list as the sum.

Output: `3`