

Solutions

3.1 Lists and Slicing

Reference example

```
values = [2, 4, 6, 8]
copy = values[1:3]
copy.append(10)
print(values)
print(copy)
```

3.1.1 - Trace

Determine the exact output and explain the state changes.

```
values = [1, 2, 3, 4, 5]
print(values[::2])
print(values[::-1])
```

Output

```
[1, 3, 5]
[5, 4, 3, 2, 1]
```

A step of 2 selects alternate positions; a negative step reverses traversal.

3.1.2 - Write code

Return a reversed copy of [3, 1, 4] while leaving the original unchanged.

Slicing with a negative step makes a new outer list.

```
values = [3, 1, 4]
result = values[::-1]
print(values)
print(result)
```

Expected output

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

Solutions

3.2 Dictionaries

Reference example

```
counts = {}
for word in ["red", "blue", "red"]:
    counts[word] = counts.get(word, 0) + 1
print(counts["red"])
print(counts.get("green", 0))
```

3.2.1 - Trace

Determine the exact output and explain the state changes.

```
data = {"a": 1, "b": 2}
data["a"] = 5
print(len(data), data["a"])
```

Output

2 5

Updating an existing key changes its value without increasing the number of keys.

3.2.2 - Write code

Build a frequency dictionary for [2, 2, 3] and print the counts in sorted key order.

Sorting the keys makes output deterministic.

```
counts = {}
for value in [2, 2, 3]:
    counts[value] = counts.get(value, 0) + 1
for key in sorted(counts):
    print(key, counts[key])
```

Expected output

2 2
3 1

Solutions

3.3 Sets and Tuples

Reference example

```
unique = set([3, 1, 3])
point = (2, 5)
print(sorted(unique))
print(point[0])
```

3.3.1 - Trace

Determine the exact output and explain the state changes.

```
a = {1, 2, 3}
b = {3, 4}
print(sorted(a & b))
print(sorted(a | b))
```

Output

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

Intersection keeps common elements; union keeps elements appearing in either set.

3.3.2 - Explain

Why is (1, 2) usable as a dictionary key while [1, 2] is not?

The tuple of integers is hashable and immutable in structure. A list is mutable and unhashable. Not every tuple is hashable if it contains unhashable elements.

Solutions

3.4 Comprehensions and Iteration Patterns

Reference example

```
values = [-2, 0, 3, 4]
squares = [value * value for value in values if value > 0]
print(squares)
```

3.4.1 - Trace

Determine the exact output and explain the state changes.

```
names = ["Ada", "Lin"]
print([f"{i}:{name}" for i, name in enumerate(names, start=1)])
```

Output

```
['1:Ada', '2:Lin']
```

enumerate supplies both a position counter and the current value.

3.4.2 - Write code

Use a comprehension to keep even values from [1, 2, 3, 4, 5] and double them.

Filter by evenness, then transform each kept value.

```
values = [1, 2, 3, 4, 5]
print([2 * x for x in values if x % 2 == 0])
```

Expected output

```
[4, 8]
```