BCA\BSc (it) Python Programming Journal - Part 2

Comprehensive Solutions for Exercises 16-26

Table of Contents

---

16. Creating List Objects

Different ways to create lists in Python:

```
# Method 1: Using square
brackets list1 = [1, 2, 3,
4, 5] print("List 1:",
list1)

# Method 2: Using list()
constructor list2 =
list(range(1, 6)) print("List
2:", list2)

# Method 3: List
comprehension list3 = [x
```

```
for x in range(1, 6)]
print("List 3:", list3) #
Method 4: From string

list4 = list("Hello")
print("List 4:", list4)

# Method 5: Mixed data types
list5 = [1, "Python", 3.14,
True] print("List 5:",
list5)
```

Output:

```
List 1: [1, 2, 3, 4, 5]
List 2: [1, 2, 3, 4, 5]
List 3: [1, 2, 3, 4, 5]
List 4: ['H', 'e', 'l', 'l',
'o'] List 5: [1, 'Python', 3.14,
True]
```

---

17. List Methods

Common list methods with examples:

```
# Create a sample list
numbers = [1, 2, 3, 4, 5,
2]

# list() - create from
iterable new_list = list((6,
7, 8))

# len() - get length
print("Length:", len(numbers))

# count() - count occurrences
print("Count of 2:",
numbers.count(2))

# index() - find position
print("Index of 4:",
numbers.index(4))

# append() - add to end
```

```
numbers.append(6)
print("After append:",
numbers)

# insert() - add at position
numbers.insert(0, 0)
print("After insert:",
numbers)

# extend() - add multiple
elements numbers.extend([7,
8]) print("After extend:",
numbers)

# remove() - delete first
occurrence numbers.remove(2)
print("After remove:", numbers)

# pop() - remove and return
element popped = numbers.pop()
print("Popped element:",
popped) print("After pop:",
numbers)

# reverse() - reverse list
numbers.reverse()
print("After reverse:",
numbers)

# sort() - sort list
numbers.sort()
print("After sort:",
numbers)

# copy() - create shallow copy
copy_list = numbers.copy()
print("Copy:", copy_list)

# clear() - empty list
numbers.clear()
print("After clear:",
numbers)
```

Output:

```
Length: 6
Count of 2: 2
```

```
Index of 4: 3
After append: [1, 2, 3, 4, 5, 2, 6]
After insert: [0, 1, 2, 3, 4, 5, 2, 6]
After extend: [0, 1, 2, 3, 4, 5, 2, 6, 7, 8]
After remove: [0, 1, 3, 4, 5, 2, 6, 7, 8]
Popped element: 8
After pop: [0, 1, 3, 4, 5, 2, 6, 7]
After reverse: [7, 6, 2, 5, 4, 3, 1, 0]
After sort: [0, 1, 2, 3, 4, 5, 6, 7]
Copy: [0, 1, 2, 3, 4, 5, 6, 7]
After clear: []
```

---

18. Creating Tuple Objects Different ways to create tuples:

```
# Method 1: Using parentheses
tuple1 = (1, 2, 3, 4)
print("Tuple 1:", tuple1)

# Method 2: Using tuple() constructor
tuple2 = tuple([1, 2, 3, 4])
print("Tuple 2:", tuple2)

# Method 3: Single element tuple
tuple3 = (5,) # Comma required
print("Tuple 3:", tuple3)

# Method 4: Without parentheses
tuple4 = 6, 7, 8
print("Tuple 4:", tuple4)

# Method 5: From range
tuple5 = tuple(range(1, 5))
print("Tuple 5:", tuple5)

# Method 6: Packing and unpacking
a, b, c = 9, 10, 11
tuple6 = a, b, c
```

```
print("Tuple 6:", tuple6)
```

Output:

```
Tuple 1: (1, 2, 3, 4)
Tuple 2: (1, 2, 3, 4)
Tuple 3: (5,)
Tuple 4: (6, 7, 8)
Tuple 5: (1, 2, 3, 4)
Tuple 6: (9, 10, 11)
```

---

19. Pattern Printing Print

various patterns:

```
# Pattern 1: Number
triangle print("Pattern
1:") for i in range(1,
6):
    print(str(i) * i)

# Pattern 2: Alphabet
pyramid print("\nPattern
2:") for i in range(65,
70):

    for j in range(65, i + 1):
        print(chr(j), end=" ")
    print()

# Pattern 3: Star
pattern print("\nPattern
3:") rows = 5 for i in
range(rows, 0, -1):
    if i > 3:
        print("*" * 5)
    else:
        print("*" * i)
```

Output:

```
Pattern 1:
1
22
333
```

```
444
4
55555

Pattern 2:
A
A B
A B C
A B C D
A B C D E

Pattern 3:
*****
*****
*****
**
*
```

---

20. Tuple Methods

Common tuple operations:

```
# Create a sample tuple
my_tuple = (1, 2, 3, 2, 4,
2, 5)

# len() - length of tuple
print("Length:", len(my_tuple))
# count() - count occurrences
print("Count of 2:",
my_tuple.count(2))

# index() - find position
print("Index of 4:",
my_tuple.index(4))

# sorted() - return sorted list
print("Sorted tuple:",
sorted(my_tuple))

# min() and max() print("Min
value:", min(my_tuple))
print("Max value:",
max(my_tuple))
```

```
# reversed() - reverse iterator
print("Reversed tuple:",
tuple(reversed(my_tuple)))

# cmp() alternative (Python
3) def cmp(a, b):
    return (a > b) - (a < b)

tuple1 = (1, 2, 3) tuple2 = (1, 2, 4)
print("Comparison result:", cmp(tuple1,
tuple2)) Output:
```

```
Length: 7
Count of 2: 3
Index of 4: 4
Sorted tuple: [1, 2, 2, 2, 3, 4, 5]
Min value: 1
Max value: 5
Reversed tuple: (5, 2, 4, 2, 3, 2, 1)
Comparison result: -1
```

---

21. Creating Set Objects

Different ways to create sets:

```
# Method 1: Using curly
braces set1 = {1, 2, 3, 4}
print("Set 1:", set1)

# Method 2: Using set()
constructor set2 = set([4, 5,
6, 7]) print("Set 2:", set2)

# Method 3: From string
set3 = set("Python")
print("Set 3:", set3)

# Method 4: Set
comprehension set4 = {x for
x in range(1, 6)} print("Set
4:", set4)
```

```
# Method 5: Empty set empty_set = set()
print("Empty set:", empty_set,
type(empty_set)) Output:
```

```
Set 1: {1, 2, 3, 4}
Set 2: {4, 5, 6, 7}
Set 3: {'y', 'o', 't', 'n', 'h', 'P'}
Set 4: {1, 2, 3, 4, 5}
Empty set: set() <class 'set'>
```

---

22. Set Methods

Common set operations:

```
# Create sample sets
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# add() A.add(5)
print("After add:",
A)

# update()
A.update([6, 7])
print("After
update:", A)

# copy()
C = A.copy() print("Copy:", C)

# pop() popped = A.pop()
print("Popped element:",
popped) print("Set after
pop:", A)

# remove()
A.remove(3)
print("After
remove:", A)

# discard()
A.discard(10) # No error if not present
print("After discard:", A)
```

```
# clear() B.clear()
print("After
clear:", B)

# union()
print("Union:", A |
C)

# intersection()
print("Intersection:", A &
C)

# difference()
print("Difference:", A - C)
```

Output:

```
After add: {1, 2, 3, 4, 5}
After update: {1, 2, 3, 4, 5, 6, 7}
Copy: {1, 2, 3, 4, 5, 6, 7}
Popped element: 1
Set after pop: {2, 3, 4, 5, 6, 7}
After remove: {2, 4, 5, 6, 7}
After discard: {2, 4, 5, 6, 7}
After clear: set()
Union: {2, 3, 4, 5, 6, 7}
Intersection: {2, 4, 5, 6, 7}
Difference: set()
```

---

23. Creating Dictionary Objects Different

ways to create dictionaries:

```
# Method 1: Using curly braces
dict1 = {'name': 'Alice',
'age': 25} print("Dict 1:",
dict1)

# Method 2: Using dict()
constructor dict2 =
dict(name='Bob', age=30)
print("Dict 2:", dict2)
```

```
# Method 3: From list of tuples dict3 =
dict([('name', 'Charlie'), ('age', 35)])
print("Dict 3:", dict3)

# Method 4: Dictionary
comprehension dict4 = {x: x**2 for
x in range(1, 6)} print("Dict 4:",
dict4)

# Method 5: Using zip keys
= ['a', 'b', 'c'] values =
[1, 2, 3] dict5 =
dict(zip(keys, values))
print("Dict 5:", dict5)

# Method 6: Empty dictionary
empty_dict = {}
print("Empty dict:",
empty_dict)
```

Output:

```
Dict 1: {'name': 'Alice', 'age': 25}
Dict 2: {'name': 'Bob', 'age': 30}
Dict 3: {'name': 'Charlie', 'age': 35}
Dict 4: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
Dict 5: {'a': 1, 'b': 2, 'c': 3}
Empty dict: {}
```

---

24. Dictionary Methods

Common dictionary operations:

```
# Create sample dictionary
person = {'name': 'Alice', 'age': 25, 'city': 'Paris'}

# dict() - create new
dictionary new_dict =
dict(country='France')

# len() - number of keys
print("Length:", len(person))

# clear() - remove all
items temp =
person.copy()
```

```
temp.clear()
print("After clear:",
temp) # get() - access
value
print("Get name:", person.get('name'))
print("Get occupation:", person.get('occupation', 'Not
specified'))

# pop() - remove key
age = person.pop('age')
print("Popped age:",
age) print("After
pop:", person)

# popitem() - remove last
item item = person.popitem()
print("Popped item:", item)
print("After popitem:",
person)

# keys() - view keys
person = {'name': 'Alice', 'age': 25, 'city': 'Paris'}
print("Keys:", list(person.keys()))

# values() - view values
print("Values:",
list(person.values()))

# items() - view key-value pairs
print("Items:", list(person.items()))

# copy() - shallow copy
person_copy = person.copy()
print("Copy:", person_copy)

# update() - merge
dictionaries
person.update({'occupation': 'Engineer',
'age': 26}) print("After update:", person)
```

Output:

```
Length: 3
After clear: {}
Get name: Alice
Get occupation: Not specified
```

```
Popped age: 25
After pop: {'name': 'Alice', 'city': 'Paris'}
Popped item: ('city', 'Paris')
After popitem: {'name': 'Alice'}
Keys: ['name', 'age', 'city']
Values: ['Alice', 25, 'Paris']
Items: [('name', 'Alice'), ('age', 25), ('city', 'Paris')]
Copy: {'name': 'Alice', 'age': 25, 'city': 'Paris'}
After update: {'name': 'Alice', 'age': 26, 'city': 'Paris', 'occupation':
'Engineer'}
```

---

25. Returning Multiple Values Return

multiple values from a function:

```
def calculate(a, b): add = a + b
    subtract = a - b multiply = a * b
    divide = a / b if b != 0 else
    "Undefined" return add, subtract,
    multiply, divide

# Call the function
results =
calculate(10, 5)

# Unpack the results
sum_result, diff_result, prod_result, div_result = results

print(f"Addition: {sum_result}")
print(f"Subtraction: {diff_result}")
print(f"Multiplication: {prod_result}")
print(f"Division: {div_result}")

# Direct unpacking a, s, m, d =
calculate(20, 4) print(f"\n20 and
4: {a}, {s}, {m}, {d}") Output:
```

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0

20 and 4: 24, 16, 80, 5.0
```

---

26. Local vs Global Variables

Demonstrate variable scope:

```
# Global variable
global_var = "I'm
global"

def test_scope(): #
    Local variable
    local_var = "I'm
    local"
    print("\nInside
    function:")
    print(global_var) #
    Access global
    print(local_var) #
    Access local

    # Modify global with
    keyword global global_var
    global_var = "Modified
    global"

    # Create new local with same
    name new_local = "New local"
    print("New local:",
    new_local)

# Call the function
test_scope()

print("\nOutside function:")
print(global_var) # Shows modified
value

# Try to access local variable (will cause
error) try: print(local_var)
except NameError as e: print("Error
accessing local_var:", e)
```

Output:

```
Inside function:
I'm global
I'm local New
local: New local
```

```
Outside function:
Modified global
Error accessing local_var: name 'local_var' is not defined
```
