quiz-set

September 16, 2024

0.1 QUIZ - Set

Q 1:

Define a function named **create_set_and_add**.

It will take a list as parameter.

First it will create a set containing the elements below:

“Apple”, “Banana”, “Cherry”, “Orange”

Then it will the items of the parameter list into this set.

And finally it will return the final set.

Hints: * {} * add()

[1]:

```python
# Q 1:

# ---- your solution here ----
def create_set_and_add(list_to_add):
    fruit_set = {"Apple", "Banana", "Cherry", "Orange"}
    for item in list_to_add:
        fruit_set.add(item)
    return fruit_set

# call the function you defined
list_to_add = ['Cranberry', 'Grape', 'Pineapple', 'Mango']
all_fruits = create_set_and_add(list_to_add)
print(all_fruits)
```

```
{'Mango', 'Banana', 'Pineapple', 'Cherry', 'Orange', 'Cranberry', 'Apple',
'Grape'}
```

Q 2:

Define a function named **create_set_and_add_all_at_once**.

It will take a list as parameter.

First it will create a set containing the elements below:

“Apple”, “Banana”, “Cherry”, “Orange”

Then it will the items of the parameter list into this set (all at once).

And finally it will return the final set.

Hints: * set() * update()

[2]:

```
# Q 2:

# ---- your solution here ----
def create_set_and_add_all_at_once(list_to_add):
    fruit_set = {"Apple", "Banana", "Cherry", "Orange"}
    fruit_set.update(list_to_add)
    return fruit_set


# call the function you defined
list_to_add = ['Cranberry', 'Grape', 'Pineapple', 'Mango', 'Another Fruit']
all_fruits = create_set_and_add_all_at_once(list_to_add)
print(all_fruits)
```

```
{'Mango', 'Banana', 'Pineapple', 'Another Fruit', 'Cherry', 'Orange',
'Cranberry', 'Apple', 'Grape'}
```

Q 3:

Define a function named **same_elements**.

It will take two Sets as parameters.

The function will return the same elements (intersection) of both sets in a List.

And this list is going to be sorted in ascending order.

Hints: * no loops * intersection() * sorted()

[3]:

```
# Q 3:

# ---- your solution here ----
def same_elements(set_1, set_2):
    return sorted(set_1.intersection(set_2))

# call the function you defined
set_1 = {10, 20, 30, 40, 50, 60}
set_2 = {20, 40, 60, 80, 90, 100}
intersection = same_elements(set_1, set_2)
print(intersection)
```

```
[20, 40, 60]
```

Q 4:

Define a function named **all_elements**.

It will take two Sets as parameters.

The function will return all the elements (union) of both sets in a List.

And this list is going to be sorted in ascending order.

Hints: * no loops * union() * sort()

[4]:

```python
# Q 4:

# ---- your solution here ----
def all_elements(set_1, set_2):
    return sorted(set_1.union(set_2))

# call the function you defined
set_1 = {10, 20, 30, 40, 50, 60}
set_2 = {20, 40, 60, 80, 90, 100}
union = all_elements(set_1, set_2)
print(union)
```

```
[10, 20, 30, 40, 50, 60, 80, 90, 100]
```

Q 5:

Define a function named **get_difference**.

It will take two lists (list_1, list_2) as parameters.

The function will return the Set of elements which are in list_1 but not in list_2.

Hints: * no loops * set() * difference()

[5]:

```python
# Q 5:

# ---- your solution here ----
def get_difference(list_1, list_2):
    return set(list_1).difference(set(list_2))

# call the function you defined
l_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
l_2 = [2, 4, 6, 8]
diff = get_difference(l_1, l_2)
print(diff)
```

```
{1, 3, 5, 7, 9}
```

Q 6:

Define a function named **is_completely_different**.

It will check if two sets (parameters) are completely different or not.

Completely different means they have no elements in common.

- If they are completely different it will return -> "They are completely different."
- If they have any elements in common it will return -> "They are not completely different."

The function will also check if the two parameters are Set or not.

If any of them is not a Set it will raise an Exception as "Parameters must be of Set type."

Hints: * no loops * completely different: `isdisjoint()` * isinstance() * raise Exception()

[6]:

```python
# Q 6:

# ---- your solution here ----
def is_completely_different(set_1, set_2):
    if not isinstance(set_1, set) or not isinstance(set_2, set):
        raise Exception("Parameters must be of Set type.")

    if set_1.isdisjoint(set_2):
        return "They are completely different."
    else:
        return "They are not completely different."

# call the function you defined
set_1 = {20, 10, 40, 30, 50}
set_2 = {60, 80, 70, 100, 90}
print(is_completely_different(set_1, set_2))

set_1 = {20, 10, 40, 30, 50, 60}
set_2 = {60, 80, 70, 90, 40, 10}
print(is_completely_different(set_1, set_2))
```

```
They are completely different.
They are not completely different.
```

Q 7:

Define a function named **is_completely_different_2**.

It will check if two sets (parameters) are completely different or not.

Completely different means they have no elements in common.

- If they are completely different it will return -> "They are completely different."
- If they have any elements in common it will return -> "They are not completely different:"
  - And it will give the common elements in a set

The function will also check if the two parameters are Set or not.

If any of them is not a Set it will raise an Exception as "Parameters must be of Set type."

Hints: * no loops * completely different: `isdisjoint()` * isinstance() * raise Exception()

[7]:

```
# Q 7:

# ---- your solution here ----
def is_completely_different_2(set_1, set_2):
    if not isinstance(set_1, set) or not isinstance(set_2, set):
        raise Exception("Parameters must be of Set type.")

    common_elements = set_1.intersection(set_2)

    if set_1.isdisjoint(set_2):
        return "They are completely different."
    else:
        return f"They are not completely different: {common_elements}"

# call the function you defined
set_1 = {20, 10, 40, 30, 50}
set_2 = {60, 80, 70, 100, 90}
print(is_completely_different_2(set_1, set_2))

set_1 = {20, 10, 40, 30, 50, 60}
set_2 = {60, 80, 70, 90, 40, 10}
print(is_completely_different_2(set_1, set_2))
```

```
They are completely different.
They are not completely different: {40, 10, 60}
```

Q 8:

Define a function named **copy_and_clear**.

It will take a Set as parameter.

And it will clear (remove all elements) this Set but copy its elements into another Set.

Finally it will return this new Set.

Hints: * no loops * copy() * clear() * remember Pass by Reference

[8]:

```
# Q 8:

# ---- your solution here ----
def copy_and_clear(original_set):
    copied_set = original_set.copy()
    original_set.clear()
    return copied_set


# call the function you defined
set1 = {'A', 'B', 'C', 'D', 'E'}
print('before the function call -> set1:', set1)
set1_copy = copy_and_clear(set1)
print('after the function call -> set1:', set1)
print('set1_copy:', set1_copy)
```

```
before the function call -> set1: {'A', 'B', 'C', 'E', 'D'}
after the function call -> set1: set()
set1_copy: {'A', 'B', 'C', 'E', 'D'}
```

Q 9:

Define a function named **remove_common_elements**.

It will take two Sets (set_1, set_2) as parameters.

The function will remove the elements of set_1 which are also in set_2.

And this operation will change the original sets.

Namely it will mutate the parameter.

Hints: * no loops * difference_update()

[10]:

```
# Q 9:

# ---- your solution here ----
def remove_common_elements(set_1, set_2):
    set_1.difference_update(set_2)

# call the function you defined
set_1 = {'a', 'b', 'c', 'd', 'e', 'f'}
set_2 = {'d', 'b', 'e', 'f', 'h', 'g'}
print('before the function -> set_1:', set_1)
remove_common_elements(set_1, set_2)
print('after the function -> set_1:', set_1)
```

```
before the function -> set_1: {'d', 'b', 'c', 'e', 'a', 'f'}
after the function -> set_1: {'c', 'a'}
```

Q 10:

Define a function named **which_one_is_superset**.

It will take two Sets (set_1, set_2) as parameters.

If set_1 is the superset of set_2 it will return: * “set_1: {….} is superset of set_2: {….}”

It the opposite is true it will return: * “set_2: {….} is superset of set_1: {….}”

Here “{….}” are set elements.

Hints: * no loop * issuperset()

[11]:

```python
# Q 10:

# ---- your solution here ----
def which_one_is_superset(set_1, set_2):
    if set_1.issuperset(set_2):
        return f"set_1: {set_1} is superset of set_2: {set_2}"
    elif set_2.issuperset(set_1):
        return f"set_2: {set_2} is superset of set_1: {set_1}"

# call the function you defined
set_1 = {'a', 'b', 'c', 'd'}
set_2 = {'d', 'b', 'e', 'f', 'a', 'c'}
print(which_one_is_superset(set_1, set_2))

set_1 = {'d', 'b', 'e', 'f', 'a', 'c'}
set_2 = {'a', 'b', 'c', 'd'}
print(which_one_is_superset(set_1, set_2))
```

```
set_2: {'d', 'b', 'c', 'e', 'a', 'f'} is superset of set_1: {'d', 'c', 'a', 'b'}
set_1: {'d', 'b', 'c', 'e', 'a', 'f'} is superset of set_2: {'d', 'c', 'a', 'b'}
```

[ ]: