Sets and Dictionaries

Exercises

Week 7

Prior to attempting these exercises ensure you have read the lecture notes and/or viewed the video, and followed the practical. You may wish to use the Python interpreter in interactive mode to help work out the solutions to some of the questions.

Download and store this document within your own filespace, so the contents can be edited. You will be able to refer to it during the test in Week 6.

Enter your answers directly into the highlighted boxes.

For more information about the module delivery, assessment and feedback please refer to the module within the MyBeckett portal.

Specify two ways in which a Set varies from a List.

Answer:

Unlike List elements, Set have no inherent order hence, we cannot access them using indices. Also Set does not allow duplicate elements. If we try to add a element; the Set will remain unchanged.

Write a Python statement that uses the `set()` *constructor* to produce the same Set as the following -

```
languages = { "C++", "Java", "C#", "PHP", "JavaScript" }
```

Answer:

languages = set(["C++", "Java", "C#", "PHP", "JavaScript"])

Is a Set **mutable** or **immutable**?

Answer:

mutable. Exception for frozenset()

Why does a Set not support *indexing* and *slicing* type operations?

Answer:

Unlike lists or tuples, sets do not have a defined order for their elements. Indexing and slicing operations rely on the concept of an ordered sequence, where elements have a specific position (index) within the sequence. Hence, Set do not support indexing and slicing operations.

Why is a `frozenset()` different from a regular set?

Answer:

forzenset() is immutable while regular Set is mutable.

How many elements would exist in the following set?

```
names = set("John", "Eric", "Terry", "Michael", "Graham", "Terry")
```

Answer:

5 elements.
Output: {'John', 'Terry', 'Eric', 'Graham', 'Michael'}

And how many elements would exist in this set?

```
vowels = set("aeiou")
```

Answer:

5 elements.
Output: {'u', 'e', 'i', 'a', 'o'} , elements position may vary.

What is the name given to the following type of expression which can be used to programmatically populate a set?

```
chars = {chr(n) for n in range(32, 128)}
```

Answer:

It is referred as “set comprehension”

What **operator** can be used to calculate the intersection (common elements) between two sets?

Answer:

“&” (ampersand)

What **operator** can be used to calculate the difference between two sets?

Answer:

“-”(minus)

What would be the result of each of the following expressions?

```
{ "x", "y", "z" } < { "z" , "u", "t", "y", "w", "x" }
```

Answer:

True , because the first set is a proper subset of the second set.

```
{ "x", "y", "z" } < { "z", "y", "x" }
```

Answer:

False , because the first set is not a proper subset of the second set and they are equal.

```
{ "x", "y", "z" } <= { "y", "z", "x" }
```

Answer:

True , because the first set is a proper subset of the second set.

```
{ "x" } > { "x" }
```

Answer:

False, because the first set is not a proper superset of the second set and they are equal.

```
{ "x", "y" } > { "x" }
```

Answer:

True , because the first set is a proper subset of the second set.

```
{ "x", "y" } == { "y", "x" }
```

Answer:

True , because the sets are equal and the order doesn't matter in sets.

Write a Python statement that uses a **method** to perform the equivalent of the following operation -

```
languages = languages | { "Python" }
```

Answer:

languages = languages.union({"Python"})

Do the elements which are placed into a set always remain in the same position?

Answer:

No, the elements in a set in Python do not have a specific order, and their positions are not guaranteed to remain the same.

Is the following operation a **mutator** or an **accessor**?

```
languages &= oo_languages
```

Answer:

mutator operation

What term is often used to refer to each *pair* of elements stored within a **dictionary**?

Answer:

“key-value-pair”

Is it possible for a dictionary to have more than one **key** with the same value?

Answer:

Yes, it is possible for a dictionary to have more than one key with the same value. However, each key in a dictionary must be unique.
Eg: my_dict = {'name': 'Monika', 'age': 22, 'city': ‘Kathmandu', bff_name: 'Monika'}

Is it possible for a dictionary to have the same **value** appear more than once?

Answer:

Yes, it is possible for a dictionary to have the same value appear more than once. Values in a dictionary can be duplicated.

Is a Dictionary **mutable** or **immutable**?

Answer:

mutable

Are the **key** values within a dictionary **mutable** or **immutable**?

Answer:

immutable

How many *elements* exist in the following dictionary?

```
stock = {"apple":10, "banana":15, "orange":11}
```

Answer:

3

And, what is the data-type of the **keys**?

Answer:

“str”

And, what output would be displayed by executing the following statement -

```
print(stock["banana"])
```

Answer:

15

Write a Python statement that uses the `dictionary()` *constructor* to produce the same dictionary as the following -

```
lang_gen = { "Java":3, "Assembly":2, "Machine Code":1 }
```

Answer:

lang_gen = dict(Java=3, Assembly=2, Machine_Code=1)

Now write a simple expression that tests whether the word `"Assembly"` is a member of the dictionary.

Answer:

print("Assembly" in lang_gen)

Write some Python code that uses a `for` statement to iterate over a dictionary called `module_stats` and print only its **values** (i.e. do not output any keys) -

Answer:

```
module_stats = {'jhon_wick': 269, 'thanos': 99999, 'spider_man': 0}

for value in module_stats.values():
    print(value)
```

Now write another loop which prints the only the **keys** -

Answer:

```
module_stats = {'jhon_wick': 269, 'thanos': 99999, 'spider_man': 0}

for value in module_stats.keys():
    print(value)
```

Is it possible to construct a dictionary using a **comprehension** style expression, as supported by lists and sets?

Answer:

Yes, it is very similar to set comprehension, but generates both **key** and a **value**.

When a Dictionary type value is being passed as an argument to a function, what characters can be used as a prefix to force the dictionary to be **unpacked** prior to the call being made?

Answer:

** (double star) operator as a prefix can be used to force the dictionary to be unpacked prior to call being made.

Exercises are complete

Save this logbook with your answers. Then ask your tutor to check your responses to each question.