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:

The two ways in which a Set varies from a List are, **Unique elements and Mutabilty.**

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

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

Answer:

A Set is **mutable**.

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

Answer:

A set in does not support indexing and slicing operations because sets are inherently unordered collections of unique elements.

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

Answer:

frozenset() is different from a regular set because, a regular Set is mutable, allowing dynamic modifications, while a **frozenset is immutable**.

How many elements would exist in the following set?

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

Answer:

Five elements would exist in the following set.

And how many elements would exist in this set?

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

Answer:

There are **five** unique vowels in the set.

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:

The name given to the following type of expression which can be used to programmatically populate a set is **set comprehension**.

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

Answer:

The operator used to calculate the intersection (common elements) between two sets is the **"&" (ampersand).**

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

Answer:

The operator used to calculate the difference between two sets is the **"-" (minus/hyphen)** operator.

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

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

Answer:

True

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

Answer:

False

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

Answer:

True

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

Answer:

False

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

Answer:

True

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

Answer:

True

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 remain in the same position. Sets are an unordered collection of unique elements.

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

```
languages &= oo_languages
```

Answer:

The operation languages &= oo_languages is a **mutator.**

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

Answer:

The term used to refer to each pair of elements stored within a dictionary is **"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.

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 more than one key with the same value.

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

Answer:

A Dictionary is **mutable.**

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

Answer:

The key values within a dictionary is **immutable.**

How many *elements* exist in the following dictionary?

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

Answer:

The dictionary stock contains **3 elements.**

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

Answer:

The data type of the keys in the dictionary is **string**.

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:

```
assembly_in_dict = "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 = {'math': 95, 'english': 88, 'history': 75, 'science': 92}

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

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

Answer:

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

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

Answer:

Yes, it is possible to construct a dictionary using a comprehension style expression known as a dictionary comprehension.

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:

The **double-asterisks (**)** can be used as a prefix to force a dictionary to be unpacked prior to being passed as an argument to a function.

Exercises are complete

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