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:

List is an ordered collection of elements, whereas set is an unordered collection of elements. List can contain duplicate elements, whereas set can not.

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 = { "C++", "Java", "C#", "PHP", "JavaScript" }
>>> new_set = set(languages)
>>> print(new_set)
{'PHP', 'C#', 'C++', 'JavaScript', 'Java'}
```

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

Answer:

Set is mutable.

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

Answer:

Set does not support indexing and slicing type operations because set is an unordered collection of elements.

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

Answer:

A frozenset() is different from a regular set because when it is created it's contents cannot be modified.

How many elements would exist in the following set?

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

Answer:

Error

And how many elements would exist in this set?

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

Answer:

5 elements would exist.

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:

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

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

Answer:

Ampersand(&) **operator** can be used to calculate the intersection (common elements) between two sets.

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

Answer:

Minus(-) **operator** can be used to calculate the difference between two sets.

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 | { "Python" }
>>> languages = languages.union({"Python"})
>>> languages
{'PHP', 'Python', 'C#', 'C++', 'JavaScript', 'Java'}
```

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

Answer:

The elements which are placed into a set do not always remain in the same position.

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

```
languages &= oo_languages
```

Answer:

Mutator

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

Answer:

key-value pair term is often used to refer to each *pair* of elements stored within a **dictionary.**

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

Answer:

It is not 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:

It is not possible for a dictionary to have the same **value** appear more than once.

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

Answer:

Dictionary is mutable.

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

Answer:

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

How many *elements* exist in the following dictionary?

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

Answer:

3 *elements* exist.

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

Answer:

<class 'dict'>

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 = { "Java":3, "Assembly":2, "Machine Code":1 }
>>> lang_gen = dict({"Java":3, "Assembly":2, "Machine Code":1})
>>> lang_gen
{'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:

```
>>> lang_gen = {"Java":3, "Assembly":2, "Machine Code":1}
>>> if "Assembly" in lang_gen:
...     print("The word 'Assembly' is a member of the dictionary.")
... else:
...     print("The word 'Assembly' is not a member of the dictionary.")
...
The word 'Assembly' is a member of the dictionary.
```

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 = {"Java":3, "Assembly":2, "Machine Code":1}
>>>
>>> for value in module_stats.values():
...     print(value)
...
3
2
1
```

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

Answer:

```
>>> module_stats = {"Java":3, "Assembly":2, "Machine Code":1}
>>>
>>> for key in module_stats.keys():
...     print(key)
...
Java
Assembly
Machine Code
```

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, as supported by lists and sets.

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

Exercises are completed