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:

Set is unordered and does not allow duplicates whereas list is ordered and can contain duplicated elements.

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" })
 Set1=set(languages)
```

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

Answer:

mutable

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

Answer:

Set does not support indexing and slicing because of its unordered nature of collection.

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

Answer:

The regular set is mutable, we can add, remove, or modify elements of the set after it is created whereas frozenset() is immutable, we cannot change its elements after creation, it is more like read only versions of set().

How many elements would exist in the following set?

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

Answer:

typeerror

And how many elements would exist in this set?

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

Answer:

5

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

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

Answer:

&
Or .intersection() method

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

Answer:

-

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.add("Python")

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

Answer:

No

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 pairs

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. In Python dictionaries, each key must be unique, but multiple keys can map to the same value.

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

Answer:

Yes it is possible.

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

Answer:

Dictionary is mutable.

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

Answer:

Key is immutable whereas values can be mutable as well as 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:

"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": 90, "Science": 85, "English": 88}

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

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

Answer:

```
module_stats = {"Math": 90, "Science": 85, "English": 88}

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.
{key_expression: value_expression for item in iterable}

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:

We can use ** before the dictionary variable name

Exercises are complete

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