5. Collections

More Data Types

(Additional Resources)

ICTAcademy

INFORMATION AND COMMUNICATIONS TECHNOLOGY

Recap/1

COLLECTIONS:

- Lists [1, 2, 1, 3, “a”] — insertion order
- Sets {1, 3, 2, “a”} — unordered
- Dictionary {“a”: 1, “b”: 2} — key -> value

INDEXING:

mylist[0] -> first element

mylist[-1] -> last element

mydictionary[“key”] -> value associated with the specified key

NOTE:

Accessing elements in a set requires looping through the elements.

Recap/2

MODIFYING:

mylist.append(“new”) -> adds “new” element

mylist.remove(“new”) -> removes “new” element

mylist.pop(4) -> removes element associated with the index 4

myset.add(“new”) -> adds “new” element

myset.remove(“new”) -> removes “new” element

myset.pop() -> removes the first element

mydictionary[“new”] = value -> adds or updates the value associated with the key “new”

mydictionary.pop(“new”) -> removes the specified key and its associated value

Additional resources: List extension

list.extend(iterable): adds all the elements from the specified iterable (e.g., list, tuple, set, or dictionary) to the end of the list, extending the list.

firstlst: list[int] = [1, 5, 7, 8]

secondlst: list[int] = [2, 3, 7, 6]

1st METHOD (this method modifies the original list in place):

- **firstlst.extend(secondlst)** results in:

firstlst is [1, 5, 7, 8, 2, 3, 7, 6]

2nd METHOD (this method creates a new extended list):

- **thirdlst: list[int]** = **firstlst** + **secondlst** results in:

thirdlst is [1, 5, 7, 8, 2, 3, 7, 6]

Additional resources: Set update

set.update(iterable): adds all elements from the specified iterable (e.g., list, tuple, set, or dictionary) to the set, effectively updating the original set with new elements.

firstset: set[int] = {1, 5, 7, 8}

secondset: set[int] = {2, 3, 7, 6}

firstset.update(secondset) results in:

firstset is {1, 2, 3, 5, 6, 7, 8}

This method modifies the original set in place.

Additional resources: Exploring Sets

Sets in Python are unordered collections of unique elements. To access or iterate through the elements of a set, you must use a loop, typically a for loop, since sets do not support indexing or slicing due to their unordered nature.

$$\text{myset: set[int]} = \{5, 4, 8, 10\}$$

```
for value in myset:
    print(value)
```

results in:

5

4

8

10

Additional resources: Exploring Dictionaries/1

dictionary.items(): returns a view of the dictionary's items (key-value pairs) as tuples. This view reflects changes to the dictionary, providing a dynamic and direct way to iterate over both keys and values.

mydict: dict[str, int] = {"a": 5, "b": 2}

tuple

```
for key, value in mydict.items():
    print(key, value)
```

results in:

a 5

b 2

Additional resources: Exploring Dictionaries/2

dictionary.values(): returns a view of the dictionary's values. This view is dynamic and reflects changes to the dictionary, allowing for efficient iteration over values.

mydict: dict[str, int] = {“a”: 5, “b”: 2}

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

results in:

5

2

Additional resources: Exploring Dictionaries/3

dictionary.keys(): returns a view of the dictionary's keys. This view is dynamic and reflects changes to the dictionary, allowing for efficient iteration over keys.

mydict: dict[str, int] = {"a": 5, "b": 2}

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

results in:

a

b