PROGRAMMING PYTHON

Lecture # 7 - Collectables continued.

Code Camp | Alpha

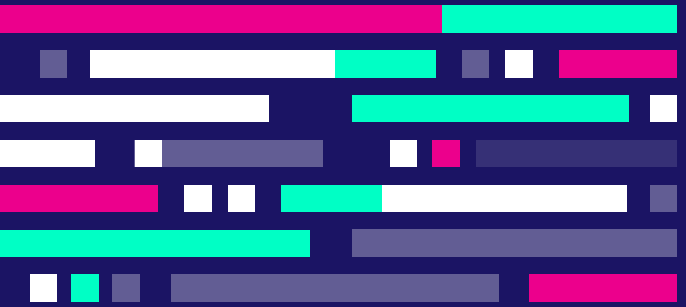
Tuples

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.
- <u>Use Cases:</u>
  - Tuples are immutable, meaning their elements cannot be changed or modified after creation. This immutability can be beneficial in situations where you want to ensure that the data remains constant throughout its lifecycle.
  - Tuples are generally faster than lists when it comes to iteration. This is because tuples are simpler and require less overhead compared to lists.
  - Tuples can be used as keys in dictionaries, while lists cannot. This is because dictionaries require keys to be immutable, and tuples satisfy that requirement.
  - Tuples can store a collection of different types of elements, just like lists. This can be useful when representing a fixed structure where each element has a specific meaning.
  - `person_info = ('John', 30, 'Male')`
  - Tuples can be used for sequence unpacking, which allows you to assign multiple variables in a single line.
  - `coordinates = (3, 4)`
  - `x, y = coordinates`

Code Camp | Alpha

Tuples

- Tuples also allow duplicates.
- We can also use tuple constructor
  - thistuple = tuple(("apple", "banana", "cherry"))
- We cannot update tuples, but by converting them into lists, updating them, and converting back to tuples. We can do so
- Count() and index() are important methods in tuples
  - Count() - Returns the number of times a specified value occurs in a tuple
  - Index() - Searches the tuple for a specified value and returns the position of where it was found.



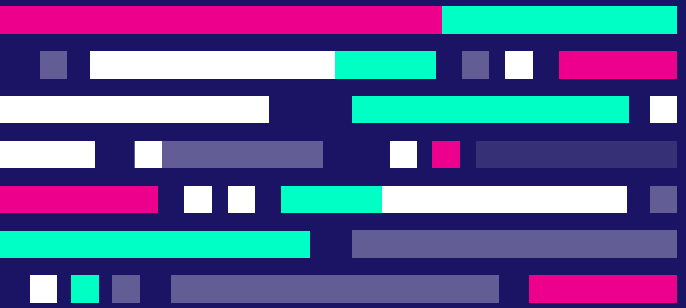
Code Camp | Alpha

Sets

- Sets are used to store multiple items in a single variable.
- A set is a collection which is unordered, unchangeable*, and unindexed.
- Note: Set items are unchangeable, but you can remove items and add new items.
- Sets are written with curly brackets.
  - thisset = {"apple", "banana", "cherry"}
- Sets are unordered, so you cannot be sure in which order the items will appear.
- Cannot be referred to by index or key.
- Use Cases:
  - Sets automatically enforce uniqueness, so you can quickly eliminate duplicate elements from a list or another iterable by converting it to a set.
  - original_list = [1, 2, 2, 3, 4, 4, 5]
  - unique_elements = set(original_list)
  - Sets support operations like union, intersection, difference, and symmetric difference, which are useful in mathematical contexts.
  - set1 = {1, 2, 3, 4}
  - set2 = {3, 4, 5, 6}
  - # Union
  - union_set = set1 | set2
  - # Intersection
  - intersection_set = set1 & set2
  - # Difference
  - difference_set = set1 - set2
  - # Symmetric Difference
  - symmetric_difference_set = set1 ^ set2

Code Camp | Alpha

Sets



- Sets can be used to find unique elements that exist in one set but not in another.
  - set1 = {1, 2, 3, 4}
  - set2 = {3, 4, 5, 6}
  - unique_to_set1 = set1 - set2
  - unique_to_set2 = set2 - set1
- Sets are useful for quickly checking if one set is a subset or superset of another.
  - set1 = {1, 2, 3, 4}
  - set2 = {2, 3}
  - is_subset = set2.issubset(set1)
  - is_superset = set1.issuperset(set2)
- Pop() and clear() randomly pops out and clears the set respectively.
- The del keyword will delete the set.

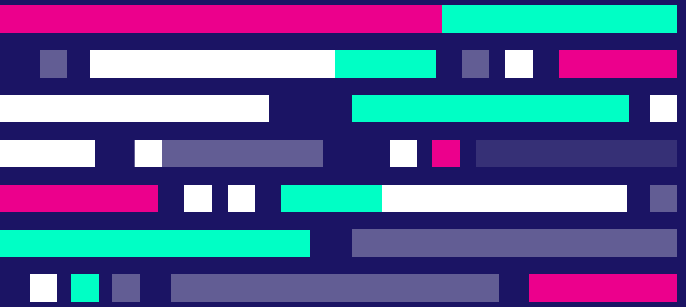
- Set Methods:
  - The union() method returns a new set with all items from both sets:
  - set1 = {"a", "b" , "c"}
  - set2 = {1, 2, 3}
  - set3 = set1.union(set2)
  - print(set3)

  The update() method inserts the items in set2 into set1:
  - set1 = {"a", "b" , "c"}
  - set2 = {1, 2, 3}
  - set1.update(set2)
  - print(set1)
  - To remove an item in a set, use the remove(), or the discard() method.
  - thisset = {"apple", "banana", "cherry"}
  - thisset.remove("banana")
  - thisset.discard("apple")

Set Methods Summary

add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

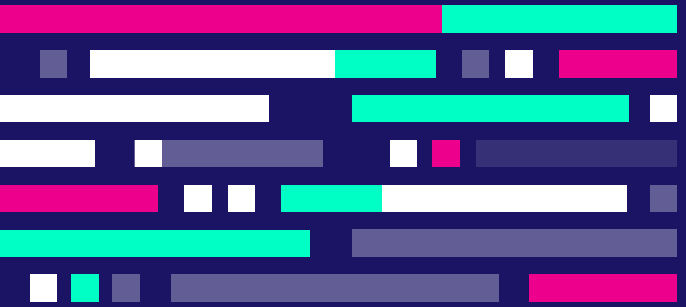
Code Camp | Alpha



Dictionary

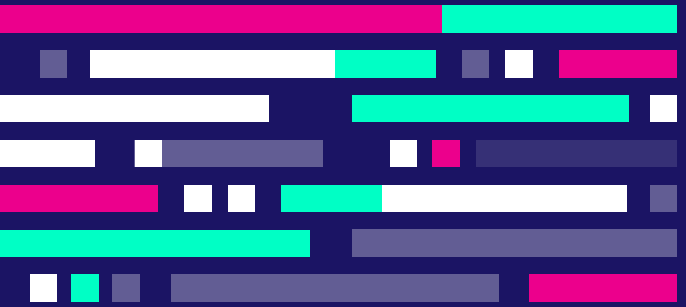
- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered, changeable and do not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values:
  - this_car = {
  - "brand": "Ford",
  - "model": "Mustang",
  - "year": 1964
  - }
- Dictionary items can be referred to by using the key name.
  - print(this_car["brand"])
- In dictionary, duplicate keys are not allowed, but values are allowed.
  - If you duplicate keys, they will be override.
- The values in dictionary can also store collectables
  - this_car = {
  - "brand": "Ford",
  - "electric": False,
  - "year": 1964,
  - "colors": ["red", "white", "blue"]
  - }

Code Camp | Alpha



Dictionary

- We can also create dictionary using its constructor function
  - person = dict(name = "]ohn", age = 36, country = "Norway")
- We can access dictionary items using their keys
  - x = person[“country"]
- There is also a method called get() that will give you the same result:
  - x = person(“country“)
- The keys() method will return a list of all the keys in the dictionary.
  - y = person.keys()
- The values() method will return a list of all the values in the dictionary.
  - y = person.values()
- We can add a key - value pair to the existing list like this:
  - person[“height”] = 45
- The items() method will return each item in a dictionary, as tuples in a list.
  - y = person.items()
- The update() method will update the dictionary with the items from the given argument.
  - person.update({"year": 2020})
- The pop() method removes the item with the specified key name:
  - person.pop(“year")



Nested Dictionary

- A dictionary can contain dictionaries, this is called nested dictionaries

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```