Python collection data types

Python Lists

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

In [ ]:
```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
    print(x)
```

In [ ]:
```
# find all elements in the list that contains the letter "a"
fruits = ["banana", "apple", "kiwi", "cherry", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

In [ ]:
```
# using list comprehension
newlist = [x for x in fruits if "a" in x]

print(newlist)
```

In [ ]:
```
import time

# Generate a large list of numbers
large_list = list(range(10000000))

# Traditional for loop
start_time = time.time()
squared_numbers_loop = []
for number in large_list:
    squared_numbers_loop.append(number**2)
loop_time = time.time() - start_time
print(f"Time taken using traditional for loop: {loop_time:.4f} seconds")

# List comprehension
start_time = time.time()
squared_numbers_comprehension = [number**2 for number in large_list]
comprehension_time = time.time() - start_time
print(f"Time taken using list comprehension: {comprehension_time:.4f} seconds")
```

In [ ]:
```
# only accept items that are not "apple"

newlist = [x for x in fruits if x != "apple"]

print(newlist)
```

In [ ]:
```
# set the values in the new list to upper case
newlist = [x.upper() for x in fruits]

print(newlist)
```

In [ ]:
```
# set all values in the new list to 'apple'
newlist = ["apple" for x in fruits]

print(newlist)
```

In [ ]:
```
# return "orange" instead of "banana"

newlist = [x if x != "banana" else "orange" for x in fruits]

print(newlist)
```

In [ ]:
```
newlist = [x for x in range(10) if x < 5]

print(newlist)
```

In [ ]:
```
mylist = fruits.copy()
mylist.sort()
print(mylist)
```

Python Tuples

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

Note: Tuples are immutable, meaning that you can't change the values in a tuple once it's created.

In [ ]:
```
fruitstuple = ("apple", "banana", "cherry", "apple", "cherry")
print(fruitstuple)
```

In [ ]:
```
print(len(fruitstuple))
```

In [ ]:
```
# one item tuple
newtuple = ("apple",)
print(type(newtuple))
```

In [ ]:
```
# tuple constructor
newtuple = tuple(("apple", "banana", "cherry"))
print(newtuple)
```

In [ ]:
```
# access tuple items is similar to list
fruitstuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(fruitstuple[2:5])
```

In [ ]:
```
# unpacking a tuple
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

In [ ]:
```
# using asterisk to assign the rest of the values to a variable

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

In [ ]:
```
for x in fruitstuple:
    print(x)
```

In [ ]:
```
# joining two tuples is similar to joining two lists
tuple1 = ("a", "b", "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

In [ ]:
```
# multiplying tuples is similar to multiplying lists
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

In [ ]:
```
# generator expression
mygenerator = (x for x in fruitstuple if "a" in x)
# mylist = [x for x in fruitstuple if "a" in x]

# generator is a special type of iterator
# the object is called when it is needed and it is not stored in memory
# so the memory is saved
print(mygenerator)

# to print all the values in the generator
print(list(mygenerator))
# or use a for loop
for x in mygenerator:
    print(x)
```

In [ ]:
```
# Generate a large list of numbers
large_list = list(range(10000000))

# List comprehension
start_time = time.time()
squared_numbers_comprehension = [number**2 for number in large_list]
comprehension_time = time.time() - start_time
print(f"Time taken using list comprehension: {comprehension_time:.4f} seconds")

# Generator expression
start_time = time.time()
squared_numbers_generator = (number**2 for number in large_list)
generator_time = time.time() - start_time
print(f"Time taken using generator expression: {generator_time:.4f} seconds")
```

Python Sets

A set is a collection which is unordered, unchangeable, and unindexed. In Python sets are written with curly brackets.

In [ ]:
```
fruitsset = {"apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"}

print(fruitsset)
print(type(fruitsset))
```

In [ ]:
```
# set constructor
myset = set(("apple", "banana", "cherry"))
print(myset)
```

In [ ]:
```
for x in fruitsset:
```

```
    print(x)
```

In [ ]:
```
print("banana" in fruitsset)
# print("watermelon" not in fruitsset)
```

In [ ]:
```
# add an item to a set
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

In [ ]:
```
# add multiple items to a set
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

In [ ]:
```
# or add any iterable to a set
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
thisset.update(mylist)
print(thisset)
```

In [ ]:
```
# union method
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

newset = thisset.union(tropical)
print(newset)
```

In [ ]:
```
newset = thisset | tropical

print(newset)
```

In [ ]:
```
# set intersection
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.intersection(set2)
print(set3)
```

In [ ]:
```
set3 = set1 & set2
print(set3)
```

In [ ]:
```
# when joining sets, the values True and 1 are considered the same
# similarly, False and 0 are also considered the same
set1 = {"apple", 1, "banana", 0, "cherry"}
set2 = {False, "google", 1, "apple", 2, True}

set3 = set1.intersection(set2)

print(set3)
```

In [ ]:
```
# remove an item from a set
# if the item does not exist, it will raise an error
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
# thisset.remove("watermelon")
print(thisset)
```

In [ ]:
```
# if the item does not exist, it will not raise an error
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
# thisset.discard("watermelon")
```

```
print(thisset)
```

In [ ]:
```
# remove the last item from the set
# since sets are unordered, the last item is arbitrary
thisset = {"apple", "banana", "cherry"}
thisset.pop()
print(thisset)
```

In [ ]:
```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

In [ ]:
```
# set difference
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.difference(set2)

print(set3)
```

In [ ]:
```
set3 = set1 - set2
print(set3)
```

In [ ]:
```
# set symmetric difference
# keep the items that are not present in both sets
set3 = set1.symmetric_difference(set2)

print(set3)
```

In [ ]:
```
set3 = set1 ^ set2
print(set3)
```

In [ ]:
```
# Generate a large list of numbers
large_list = list(range(10000000))

# Set comprehension
start_time = time.time()
squared_numbers_set = {number**2 for number in large_list}
set_time = time.time() - start_time
print(f"Time taken using set comprehension: {set_time:.4f} seconds")

# Generator expression
start_time = time.time()
squared_numbers_generator = (number**2 for number in large_list)
# To measure the time, we need to iterate through the generator
squared_numbers_generator = list(squared_numbers_generator)
generator_time = time.time() - start_time
print(f"Time taken using generator expression: {generator_time:.4f} seconds")
```

Python Dictionaries

A dictionary is a collection which is ordered, changeable and do not allow duplicates. In Python dictionaries are written with curly brackets, and they have keys and values.

In [ ]:
```
# dictionary
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
print(thisdict)
```

In [ ]:
```
# duplicate keys are not allowed
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964, "year": 2020}
print(thisdict)
```

In [ ]:
```
# the values in the dictionary can be of any data type
thisdict = {
    "brand": "Ford",
    "electric": False,
    "year": 1964,
    "colors": ["red", "white", "blue"],
}
print(thisdict)
```

In [ ]:
```
# dict constructor
thisdict = dict(
    brand="Ford",
    model="Mustang",
    year=1964,
    electric=False,
    colors=["red", "white", "blue"],
)
print(thisdict)
```

In [ ]:
```
# add a new item to the dictionary.
# see the keys list also updates
# this behavior can be seen in values() and items() methods
car = {"brand": "Ford", "model": "Mustang", "year": 1964}
x = car.keys()
print(x)  # before the change
car["color"] = "white"
print(x)  # after the change
```

In [ ]:
```
print("model" in car)
```

In [ ]:
```
# change items
# if the key does not exist, it will add the key-value pair
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
thisdict["year"] = 2018
print(thisdict)
```

In [ ]:
```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
thisdict.update({"year": 2020})
print(thisdict)
```

In [ ]:
```
# remove an item
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
thisdict.pop("model")
print(thisdict)
```

In [ ]:
```
# or use the del keyword
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
del thisdict["model"]
print(thisdict)
```

In [ ]:
```
# remove the last item
# since dictionaries are unordered, the last item is arbitrary
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
thisdict.popitem()
print(thisdict)
```

In [ ]:
```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
thisdict.clear()
print(thisdict)
```

In [ ]:
```
# loop through a dictionary using keys
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
```

```
for x in thisdict:
    print(x)

# for x in thisdict.keys():
#     print(x)
```

In [ ]:
```
# loop through a dictionary using values
for x in thisdict.values():
    print(x)
```

In [ ]:
```
# loop through both keys and values
for x, y in thisdict.items():
    print(x, y)
```

In [ ]:
```
# nested dictionaries
myfamily = {
    "child1": {"name": "Emil", "year": 2004},
    "child2": {"name": "Tobias", "year": 2007},
    "child3": {"name": "Linus", "year": 2011},
}

print(myfamily)
```

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

myfamily = {"child1": child1, "child2": child2, "child3": child3}

print(myfamily)
```

In [ ]:
```
# access the items in the nested dictionary
print(myfamily["child1"]["name"])
```

In [ ]:
```
# loop through the nested dictionary
for x, obj in myfamily.items():
    print(x)

    for y in obj:
        print(y + ":", obj[y])
```

In [64]:
```
import time
import random

# Generate a large dataset
num_students = 100000
student_ids = [random.randint(100000, 999999) for _ in range(num_students)]
student_names = [f"Student_{i}" for i in range(num_students)]

# List of tuples for student data
student_data_list = list(zip(student_ids, student_names))
# student_data_list
```

In [65]:
```
# Dictionary for fast lookups
student_data_dict = dict(zip(student_ids, student_names))
# student_data_dict
```

In [ ]:
```
# Performance test for list
start_time = time.time()
for student in student_data_list:
    if student[0] in student_ids:
        pass  # Simulate processing
```

```
list_time = time.time() - start_time
print(f"Time taken for list processing: {list_time:.4f} seconds")

# Performance test for dictionary
start_time = time.time()
for student_id in student_ids:
    if student_id in student_data_dict:
        pass  # Simulate processing
dict_time = time.time() - start_time
print(f"Time taken for dictionary processing: {dict_time:.4f} seconds")
```

In [ ]:

```
# Generate a large dataset
large_purchases = [
    {
        "customer_id": random.randint(1, 1000),
        "item": random.choice(["apple", "banana", "orange"]),
    }
    for _ in range(100000)
]

# Measure time for unique customers using a set
start_time = time.time()
unique_customers_large = {purchase["customer_id"] for purchase in large_purchases}
print(
    f"Unique customers ({len(unique_customers_large)}) (set) time: {time.time() - sta
)

# Measure time for unique customers using a dictionary
start_time = time.time()
unique_customers_dict_large = {
    purchase["customer_id"]: True for purchase in large_purchases
}
print(
    f"Unique customers ({len(unique_customers_dict_large)}) (dict) time: {time.time()
)
```