COLLECTIONS

DATA 601

PYTHON COLLECTIONS

There are four collection data types in the Python programming language:

- **List**
- **Tuple**
- **Set**
- **Dictionary**

	Symbol
Parentheses	( )
Brackets	[ ]
Braces	{ }

PYTHON COLLECTIONS: DEFINITIONS

There are four collection data types in the Python programming language:

- **List:** Ordered and changeable collection allowing duplicate members.
- **Tuple:** Ordered and unchangeable collection allowing duplicate members.
- **Set:** Unordered and unindexed collection **w/o** duplicate members.
- **Dictionary:** Unordered, changeable, and indexed collection **w/o** duplicate members.

		Ordered?	Changeable?	Indexed?	Duplicates?
List	[ ]	Yes	Yes	Yes	Yes
Tuple	( )	Yes	No	Yes	Yes
Set	{ }	No	Yes	No	No
Dictionary	{"_:_"}	No	Yes	Yes	No

LISTS

Ordered and changeable collections allowing duplicate members

HOW DO WE CREATE A LIST?

- Lists are written with square brackets.

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Output:

```
['apple', 'banana', 'cherry']
```

LOOP THROUGH A LIST

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

Output:

```
apple
banana
orange
```

CHECK IF ITEM EXISTS

```
thislist = ["apple", "banana", "cherry"]
anitem = "apple"
if anitem in thislist:
  print("Yes,", anitem,"is in the fruits list")
else:
  print("No,", anitem,"is not in the fruits list")
```

Output:

```
Yes, apple is in the fruits list
```

```
anitem = "blackberry"
if anitem in thislist:
  print("Yes,", anitem,"is in the fruits list")
else:
  print("No,", anitem,"is not in the fruits list")
```

Output:

```
No, blackberry is not in the fruits list
```

LIST LENGTH

```
thislist = ["apple", "banana", "cherry"]

print(len(thislist))
```

Output: 3

ADD ITEMS

```
thislist.append("watermelon")
print(thislist)
```

Output:

```
['apple', 'banana', 'orange', 'watermelon']
```

ADD AN ITEM AT THE SPECIFIED INDEX

```
thislist = ['apple', 'banana', 'orange', 'watermelon']
```

```
thislist.insert(1, "pear")
print(thislist)
```

Output:

```
['apple', 'pear', 'banana', 'orange', 'watermelon']
```

EXTENDING

- The extend() method adds the specified list elements (or any iterable) to the end of the current list.

```
fruits = ['apple', 'banana', 'cherry']
morefruits = ['watermelon', 'pear', 'orange', 'grape']

fruits.extend(morefruits)
print(fruits)
```

Output:

```
['apple', 'banana', 'cherry', 'watermelon', 'pear',
'orange', 'grape']
```

REMOVE AN ITEM (METHOD #1)

```
thislist=['apple', 'pear', 'banana', 'orange', 'watermelon’]

thislist.remove("banana")
print(thislist)
```

Output:

```
['apple', 'pear', 'orange', 'watermelon']
```

REMOVE AN ITEM (METHOD #2)

```
# The pop() method removes the specified index,
# (or the last item if index is not specified):

thislist = ["apple", "banana", "cherry"]

thislist.pop()

print(thislist)
```

Output:

```
['apple', 'banana']
```

CLEARING VS. DELETING A LIST

```
# The clear() method empties the list:

thislist = ["apple", "banana", "cherry"]

thislist.clear()

print(thislist)
```

Output: []

```
del thislist
```

➔ This command deletes the list.

➔ If you try to print the list, you'll get an error message

COPYING A LIST (METHOD #1)

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

```
thislist = ["apple", "banana", "cherry"]

mylist = thislist.copy()

print(mylist)
```

Output:

```
['apple', 'banana', 'cherry']
```

COPYING A LIST (METHOD #2)

```
thislist = ["apple", "banana", "cherry"]

mylist = list(thislist)

print(mylist)
```

Output:

```
['apple', 'banana', 'cherry']
```

SORTING

```
thislist = ["watermelon", "apple", "cherry", "banana"]

thislist.sort()

print(thislist)
```

Output:

```
['apple', 'banana', 'cherry', 'watermelon']
```

REVERSE ORDERING

```
thislist = ["watermelon","apple", "cherry", "banana"]

thislist.reverse()

print(thislist)
```

Output:

```
['banana', 'cherry', 'apple', 'watermelon']
```

COUNTING

```
names = ["Adam","Michael","Susan","Leo","Adam","Marry","Heather"]

names.count("Adam")
```

Output: 2

INDEXING

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

```
names =["Adam","Michael","Susan","Leo","Adam","Marry","Heather"]

names.index("Adam")
```

Output: `0`

```
names.index("Susan")
```

Output: `2`

SUMMARY

A list is a collection which is ordered and changeable.
Lists are written with square brackets.

```
append()  Adds an element at the end of the list
clear()   Removes all the elements from the list
copy()    Returns a copy of the list
count()   Returns the number of elements with the specified value
extend()  Add the elements of a list (or any iterable), to the end of the current list
index()   Returns the index of the first element with the specified value
insert()  Adds an element at the specified position
pop()     Removes the element at the specified position
remove()  Removes the item with the specified value
reverse() Reverses the order of the list
sort()    Sorts the list
```

TUPLES

Ordered & unchangeable collections allowing duplicate members

HOW DO WE CREATE A TUPLE?

- Tuples are written with round brackets.

```
thistuple = ("apple", "banana", "cherry")

print(thistuple)
```

Output:

```
('apple', 'banana', 'cherry')
```

LOOP THROUGH A TUPLE

```
thistuple = ("apple", "banana", "cherry")

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

Output:

```
apple
banana
orange
```

CHECK IF ITEM EXISTS

```
thistuple = ("apple", "banana", "cherry")
anitem = "apple"
if anitem in thistuple:
  print("Yes,", anitem,"is in this tuple")
else:
  print("No,", anitem,"is not in this tuple")
```

Output:

```
Yes, apple is in this tuple
```

```
thistuple = ("apple", "banana", "cherry")
anitem = "grape"
if anitem in thistuple:
  print("Yes,", anitem,"is in this tuple")
else:
  print("No,", anitem,"is not in this tuple")
```

Output:

```
No, grape is not in this tuple
```

TUPLE LENGTH

```
thistuple = ("apple", "banana", "cherry")

print(len(thistuple))
```

Output: 3

ADD/REMOVE ITEMS, EXTEND TUPLE SORTING AND REVERSE ORDERING

YOU CANNOT DO ANY OF THESE

TUPLES ARE UNCHANGEABLE

DELETING A TUPLE

```
del thistuple
```

➔ This command deletes the tuple.

➔ If you try to print the tuple after deleting, then you'll get an error message

COPYING A TUPLE

You cannot copy a list with the = sign because lists are mutables.
The = sign creates a reference not a copy.
Tuples are immutable therefore a = sign does not create a reference but a copy as expected.

```
thistuple = ("apple", "banana", "cherry")

newtuple = thistuple

print(newtuple)
```

Output:

```
('apple', 'banana', 'cherry')
```

Since tuples cannot be changed, why would someone copy a tuple?

COUNTING

```
names = ("Adam","Michael","Susan","Leo","Adam","Marry","Heather")

names.count("Adam")
```

Output: 2

INDEXING

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

```
names =("Adam","Michael","Susan","Leo","Adam","Marry","Heather")

names.index("Adam")
```

Output: `0`

```
names.index("Susan")
```

Output: `2`

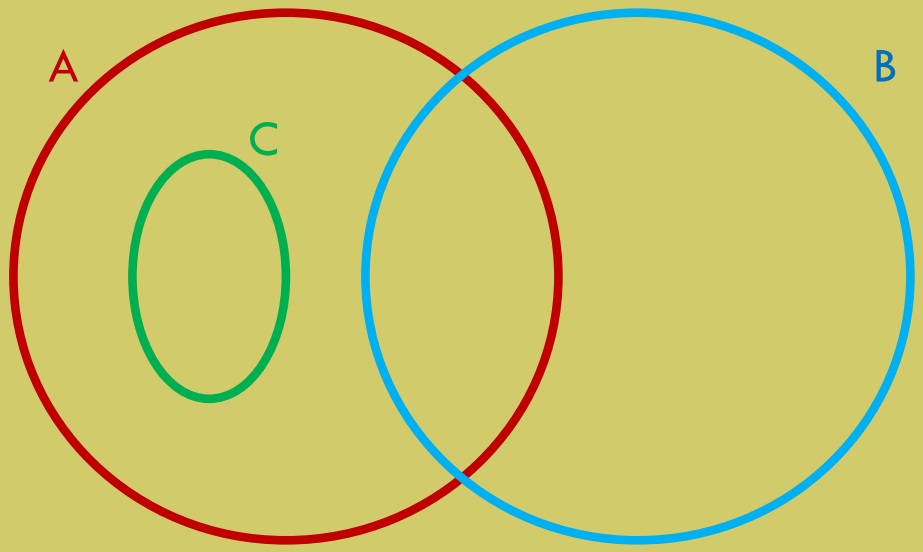
SUMMARY

Tuple is a collection which is ordered and unchangeable.

Allows duplicate members.

Tuples are written with round brackets.

```
count()   Returns the number of elements with the specified value
index()   Returns the index of the first element with the specified value
```



SETS

Unordered and unindexed collections without duplicate members

HOW DO WE CREATE A SET?

- Sets can be created with curly brackets.

```
thisset = {"apple", "banana", "cherry"}

print(thisset)
```

Output:

```
{'apple', 'cherry', 'banana'}
```

ANOTHER WAY OF CREATING SETS

Use the set() constructor to make a set.

note the double round-brackets

```
thisset = set(("apple", "banana", "cherry"))

print(thisset)
```

Output:

```
{'apple', 'cherry', 'banana'}
```

ACCESSING ITEMS

- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

What we can do?

- We can loop through the set items using a for loop
- We can ask if a specified value is present in a set, by using the in keyword.

LOOP THROUGH A SET

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:

  print(x)
```

Output:

```
apple
cherry
banana
```

CHECK IF ITEM EXISTS

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Output: `True`

```
thisset = {"apple", "banana", "cherry"}

print("grape" in thisset)
```

Output: `False`

GET LENGTH

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

Output: 3

ADD ITEMS

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Output:

```
{'apple', 'cherry', 'orange', 'banana'}
```

UPDATING

- To add more than one item to a set use the update() method.

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

Output:

```
{'cherry', 'mango', 'apple', 'orange', 'banana', 'grapes'}
```

REMOVE AN ITEM (METHOD #1)

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Output:

```
{'apple', 'cherry'}
```

REMOVE AN ITEM (METHOD #2)

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

Output: `{'apple', 'cherry'}`

What is the difference between remove and discard
If the item to remove does not exist,

- remove() will raise an error but
- discard() will NOT raise an error.

REMOVE AN ITEM (METHOD #3)

```
# Remove an item by using the pop() method:
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

Output:

```
apple
{'cherry', 'banana'}
```

NOTE THAT:
Sets are unordered, so when using the pop() method, you will not know which item that gets removed.

CLEARING VS. DELETING A SET

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

Output: `set()`

`del thisset` ➔ This command deletes the set

COPYING A SET

```
fruits = {"apple", "banana", "cherry"}
x = fruits.copy()
print(x)
```

Output:

```
{'apple', 'cherry', 'banana'}
```

FINDING THE DIFFERENCES BETWEEN TWO SETS

x.difference(y) method return a set that contains the items that only exist in set x, and not in set y:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.difference(y)
print(z)
```

Output:

```
{'cherry', 'banana'}
```

REMOVING THE ITEMS THAT EXIST IN BOTH SETS

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.difference_update(y)
print(x)
```

Output: `{'cherry', 'banana'}`

Note that
difference() method returns a new set, without the unwanted items,
difference_update() method removes the unwanted items from the original set.

FINDING THE ITEMS THAT EXIST IN TWO SETS

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y)
print(z)
```

Output: `{'apple'}`

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.intersection_update(y)
print(x)
```

Output: `{'apple'}`

Note that
intersection() method returns a new set, without the unwanted items
intersection_update() method removes the unwanted items from the original set.

MERGING SETS

union() return a set that contains all items from both sets, duplicates are excluded:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.union(y)
print(z)
```

Output:

```
{'cherry', 'google', 'banana', 'apple', 'microsoft'}
```

DETERMINING WHETHER X IS A SUBSET OF Y

issubset() teturns True if all items set x are present in set y:

```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z)
```

Output: True

DETERMINING WHETHER Y IS A SUPERSET OF X

issuperset() returns True if all items set y are present in set x:

```
x = {"f", "e", "d", "c", "b", "a"}
y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)
```

Output: True

SUMMARY

A set is an unindex and unorders collections without duplicates
Sets are written with {, , , }

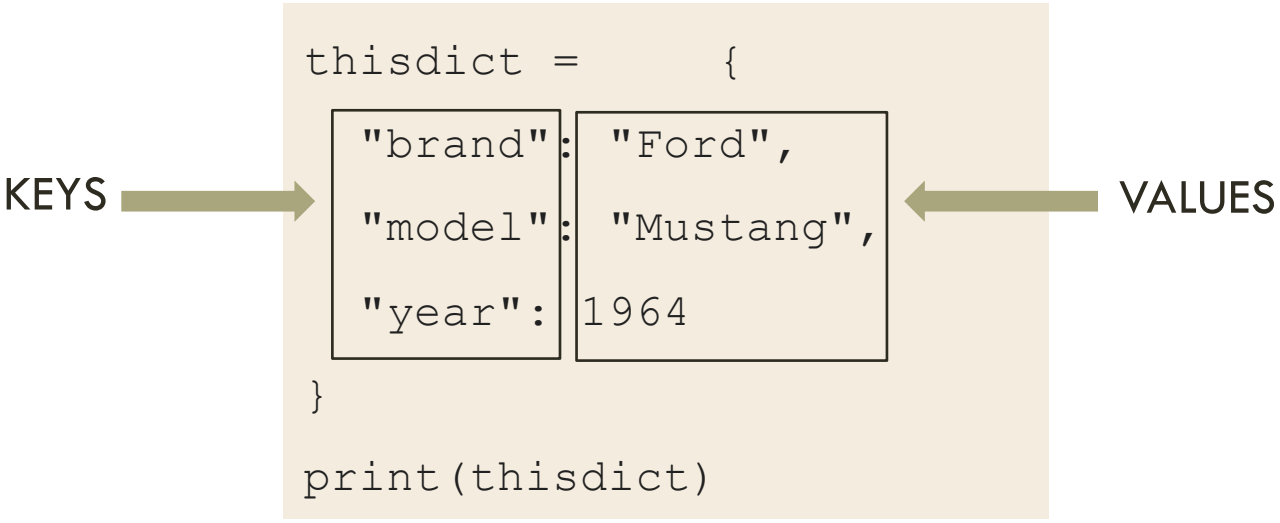
```
add()      Adds an element to the set
clear()    Removes all the elements from the set
copy()     Returns a copy of the set
update()   Adds the multiple to a set
discard()  Removes the element from the set
remove()   Removes the element from the set
union()    Merges sets
issubset() Returns 1 if subset
issuperset() Returns 1 if superset
```

DICTIONARIES

Unordered, changeable, and indexed collections without duplicate members

HOW TO CREATE A DICTIONARY

- Dictionaries are written with curly brackets
- Dictionaries have keys and values.




```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Output: `{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}`

BUILDING DICTIONARIES

Dictionary: Days

- There are multiple ways of creating dictionaries.
- Let's build a dictionary of days in English (keys) and Italian(values)
- Method-1:

```
Days_Eng_Ita = {'Monday':'Lunedi',
                'Tuesday':'Martedi',
                'Wednesday':'Mercoledi',
                'Thursday':'Geovedi',
                'Friday':'Venerdi',
                'Saturday':'Saboto',
                'Sunday':'Domenica'
               }
```

BUILDING DICTIONARIES (CONT…)

Dictionary: Days

- There are multiple ways of creating dictionaries.
- Let's build a dictionary of days in English (keys) and Italian(values)
- Method-2:

```
Days_Eng_Ita = dict([
    ('Monday','Lunedi'),
    ('Tuesday','Martedi'),
    ('Wednesday','Mercoledi'),
    ('Thursday','Geovedi'),
    ('Friday','Venerdi'),
    ('Saturday','Saboto'),
    ('Sunday','Domenica')
     ])
```

BUILDING DICTIONARIES (CONT…)

Dictionary: Days

- There are multiple ways of creating dictionaries.
- Let's build a dictionary of days in English (keys) and Italian(values)
- Method-3:

```
Days_Eng_Ita = dict(
    Monday='Lunedi',
    Tuesday='Martedi',
    Wednesday='Mercoledi',
    Thursday='Geovedi',
    Friday='Venerdi',
    Saturday='Saboto',
    Sunday='Domenica'
    )
```

BUILDING DICTIONARIES (CONT…)

Dictionary: Days

```
type(Days_Eng_Ita)
```

```
dict
```

```
Days_Eng_Ita
```

Output:

```
{'Friday': 'Venerdi',
 'Monday': 'Lunedi',
 'Saturday': 'Saboto',
 'Sunday': 'Domenica',
 'Thursday': 'Geovedi',
 'Tuesday': 'Martedi',
 'Wednesday': 'Mercoledi'}
```

BUILDING DICTIONARIES (CONT…)

Dictionary: Person

- We can even build dictionaries incrementally.
- Let's work on a new example

```
person = {}
person['fname'] = 'Jon'
person['lname'] = 'Snow'
person['age'] = 27
person['spouse'] = 'Ygritte'
person['relatives'] = ['Ned', 'Robb', 'Sansa','Arya']
person['pets'] = {'dog': 'Ghost', 'dragon': 'Drogon'}
```

BUILDING DICTIONARIES (CONT…)

Dictionary: Person

`person` → Output:

```
{'age': 27,
 'fname': 'Jon',
 'lname': 'Snow',
 'pets': {'dog': 'Ghost', 'dragon': 'Drogon'},
 'relatives': ['Ned', 'Robb', 'Sansa', 'Arya'],
 'spouse': 'Ygritte'}
```

`person['fname']` → Output: `'Jon'`

`person['relatives']` → Output: `['Ned', 'Robb', 'Sansa', 'Arya']`

`person['relatives'][0]` → Output: `'Ned'`

`person['relatives'][-1]` → Output: `'Arya'`

`person['pets']['dog']` → Output: `'Ghost'`

LOOP THROUGH A DICTIONARY

Dictionary: Person

- Print all key names in the dictionary, one by one:

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

Output:

```
fname
lname
age
spouse
relatives
pets
```

- Print all values in the dictionary, one by one:

```
for x in person:
  print(person[x])
```

Output:

```
Jon
Snow
27
Ygritte
['Ned', 'Robb', 'Sansa', 'Arya']
{'dog': 'Ghost', 'dragon': 'Drogon'}
```

LOOP THROUGH A DICTIONARY (CONT…)

Dictionary: Person

- You can also use the values() function to return values of a dictionary:

```
for x in person.values():
  print(x)
```

Output:

```
Jon
Snow
27
Ygritte
['Ned', 'Robb', 'Sansa', 'Arya']
{'dog': 'Ghost', 'dragon': 'Drogon'}
```

- Loop through both keys and values, by using the items() function:

```
for x, y in person.items():
  print(x, y)
```

Output:

```
fname Jon
lname Snow
age 27
spouse Ygritte
relatives ['Ned', 'Robb', 'Sansa', 'Arya']
pets {'dog': 'Ghost', 'dragon': 'Drogon'}
```

ANOTHER DICTIONARY EXAMPLE

Dictionary: Mustang

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

CHECK IF KEY EXISTS

Dictionary: Mustang

```
key = "model"
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if key in thisdict:
  print("Yes,", key, "is one of the keys in this dictionary")
else:
  print("No,", key, "is not one of the keys in this dictionary")
```

Output:

```
Yes, model is one of the keys in this dictionary
```

DICTIONARY LENGTH

Dictionary: Mustang

```
print(len(thisdict))
```

Output: 3

ADDING ITEMS

Dictionary: Mustang

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964,
'color': 'red'}
```

ADDING ITEMS (CONT…)

Dictionary: Mustang

We can also use update()

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 2019
}
car.update({"color": "White"})
print(car)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2019,
'color': 'White'}
```

REMOVE AN ITEM (METHOD #1)

Dictionary: Mustang

Method-1: The pop() method removes the item with the specified key name

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'year': 1964}
```

REMOVE AN ITEM (METHOD #2)

The popitem() method removes a random item!

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang'}
```

REMOVE AN ITEM (METHOD #3)

The del keyword removes the item with the specified key name:

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

Output:

```
{'brand': 'Ford', 'year': 1964}
```

Note that `del thisdict` deletes the entire dictionary

CLEARING A DICTIONARY

The clear() keyword empties the dictionary:

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

Output: {}

COPYING A DICTIONARY (METHOD #1)

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are various ways to make a copy.
One way is to use the built-in Dictionary method copy().

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

COPYING A LIST (METHOD #2)

Another way to make a copy is to use the built-in method dict().

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

CREATING A DICTIONARY WITH KEYS AND A VALUE

```
keys = {'a', 'e', 'i', 'o', 'u' }
value = 'vowel'
vowels = dict.fromkeys(keys, value)
print(vowels)
```

Output:

```
{'u': 'vowel', 'e': 'vowel', 'a': 'vowel', 'i': 'vowel', 'o': 'vowel'}
```

ALSO

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

```
thisdict.items()
```

Output:

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

```
thisdict.keys()
```

Output:

```
dict_keys(['brand', 'model', 'year'])
```

```
thisdict.values()
```

Output:

```
dict_values(['Ford', 'Mustang', 1964])
```

SUMMARY: DICT

A dictionary is a collection which is unordered, changeable and indexed.

In Python dictionaries are written with curly brackets, and they have keys and values.

`clear()`	Removes all the elements from the dictionary
`copy()`	Returns a copy of the dictionary
`fromkeys()`	Returns a dictionary with the specified keys and values
`get()`	Returns the value of the specified key
`items()`	Returns a list containing the a tuple for each key value pair
`keys()`	Returns a list containing the dictionary's keys
`pop()`	Removes the element with the specified key
`popitem()`	Removes the last inserted key-value pair
`setdefault()`	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
`update()`	Updates the dictionary with the specified key-value pairs
`values()`	Returns a list of all the values in the dictionary

SUMMARY: PYTHON COLLECTIONS

		Ordered?	Changeable?	Indexed?	Duplicates?
List	[ ]	Yes	Yes	Yes	Yes
Tuple	( )	Yes	No	Yes	Yes
Set	{ }	No	Yes	No	No
Dictionary	{"_:_"}	No	Yes	Yes	No

LIST
```
append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()
```

TUPLE
```
count()
index()
```

SET
```
add()
clear()
copy()
difference_update()
discard()
issubset()
issuperset()
len()
remove()
union()
update()
```

DICT
```
clear()
copy()
fromkeys()
get()
items()
keys()
pop()
popitem()
setdefault()
update()
values()
```