Key Term

Dictionary - An unordered collection of key-value pairs used to store data values. Dictionaries are written with curly braces {} and contain keys and values separated by a colon.

Key - The unique identifier used to access values in a dictionary. Keys can be strings, numbers, or other immutable objects.

Value - The data associated with each key. Values can be numbers, strings, lists, or other objects.

```
# Create a dictionary
dict1 = {"key1": "value1", "key2": 2}

# Access value by key
print(dict1["key1"])

# Add new key-value
dict1["key3"] = "value3"
print(dict1)
```

Run

Reset

```
value1
{'key1': 'value1', 'key2': 2, 'key3': 'value3'}
```

Example of dictionary in Bash

```
# Create a dictionary
dict1 = {"key1": "value1", "key2": 2}

# Access value by key
print(dict1["key1"])

# Add new key-value
dict1["key3"] = "value3"
print(dict1)
```

Set - An unordered collection of unique elements. Sets are written with curly braces {}

Set operations - Methods to interact with sets like union, intersection, difference and symmetric difference.

```
# Create a set
set1 = {"value1", "value2", "value3"}

# Add new element
set1.add("value4")
print(set1)

# Set union
set2 = {"value4", "value5", "value6"}
set3 = set1 | set2
print(set3)

# Set intersection
set4 = set1 & set2
print(set4)

# Set difference
set5 = set1 - set2
print(set5)
```

```
{'value2', 'value4', 'value3', 'value1'}
{'value2', 'value5', 'value6', 'value1', 'value4', 'value3'}
{'value4'}
{'value2', 'value1', 'value3'}
```

Creating Dictionaries in Python

- Dictionaries are data structures which map keys to values
- ↳ use highly efficient mechanism to look up the data using these keys. ← relies on what's known as a hash function

Creating Dictionaries

All Immutable in Python have hash functions → so can be used as a dictionary

```
[1]: key_1 = "henry"
```

↳ string ↳ strings, Tuples

```
[2]: key_1.__hash__()
```

↳ returns a predictable value, which is used in dictionary lookups ← sequence table that be mutable type cannot be used as a dictiona ↳ list and will not have hash function

```
[2]: -6626405913583883268
```

```
[5]: dict(name="Henry", age="16")
```

↓ represents as curly brackets

```
[5]: {'name': 'Henry', 'age': '16'}
```

key value key value

Three ways to create a Dictionary

```
[6]: dict([["Name","Henry"],["age",16]])
```

↳ The list of lists as an argument

```
[6]: {'Name': 'Henry', 'age': 16}
```

```
[8]: {'name':'Henry', 'age':16}
```

```
[8]: {'name': 'Henry', 'age': 16}
```

```
[9]: {'name':'Henry', 'age':16} == {'age':16, 'name': 'Henry'}
```

```
[9]: True
```

order is not count

↓

It's considered the values of the keys

that are they equal or not?

Accessing Dictionary Data in Python

- Accessing, Updating and Adding

Accessing Dictionary Values

```
[2]: student = { 'name': 'Sparky',
                 'age': 27,
                 'major': 'Basketweaving' }
```

→ similary to index in list for look up, but Dic. is use Key for look up instead number of index

```
[3]: student['age']

[3]: 27

[4]: student['dog']
```

↓ if the key does not exist

```
KeyError                                  Traceback (most recent call last)
/var/folders/29/5dyw2t0n71v13sp1980c30g80000gr/T/ipykernel_1415/96716775.py in <module>
----> 1 student['dog']

KeyError: 'dog'
```

if we don't set it will be None

```
[ ]: if 'dog' in student:
         student['dog']
```

↵ to avoid this ← This is not inwork, because of this key does not exist

default value for the incorrect key ↰

```
[9]: student.get('dog', 'MISSING KEY')

[9]: 'MISSING KEY'
```

↵ from this there's a special method 'get'

↳ alternative way of accessing values in a dictionary

↓ id the key correct, it'll show the value

```
[11]: student['major'] = 'Math'
```

← to update a value of an existing key

```
[12]: student

[12]: {'name': 'Sparky', 'age': 27, 'major': 'Math'}
```

↙ updated

```
[13]: student['grade'] = 'A'

[14]: student

[14]: {'name': 'Sparky', 'age': 27, 'major': 'Math', 'grade': 'A'}
```

→ if the pair doesn't exist it will add to dic

Dictionary Views in Python

Dictionary Views

```
[1]: student = { 'name': 'Sparky',
                 'age': 27,
                 'major': 'Math' }

[5]: student.keys()

[5]: dict_keys(['name', 'age', 'major'])

[6]: student.values()

[6]: dict_values(['Sparky', 27, 'Math'])

[8]: student.items()

[8]: dict_items([('name', 'Sparky'), ('age', 27), ('major', 'Math')])

[11]: 'name' in student.keys()

[11]: True

[13]: 'name' in student

[13]: True

[14]: for key in student:
          print(key)

name
age
major
```

↳ don't forget

↓ if we don't specify the method it'll return as keys

* * *

```
[15]: for key, value in student.items():
          print(key)
          print(value)

name
Sparky
age
27
major
Math
```

Sets and Set Operations in Python

Sets

- **Sets :** unordered collection of unique objects
  - contain items of any immutable or hashable datatype ← cannot contain mutable objects

```
[4]: set("aaabbbcc")
[4]: {'a', 'b', 'c'}
[7]: {'a', 'b', 'c', 'c'}
[7]: {'a', 'b', 'c'}
[8]: s = set('aaabbbccc')
     s
[8]: {'a', 'b', 'c'}
[9]: s.add('d')
     s
[9]: {'a', 'b', 'c', 'd'}
[10]: 'd' in s
[10]: True
[11]: s.pop()
[11]: 'd'
[12]: s.remove('b')
```

- return to unique item
- not a dic. cause there's no key value pairs
- create set by using curly brachet
- ← no bracket, use add method because set has no idea of order, so you can't index into it
- normal test
- → to remove from the set
  - ↳ random item
  - ↳ can't rely on the set returning the last item
- ↳ specific the item to remove from the set

Set Operations

Disjoint

Two sets are disjoint if they have no items in common

```
[17]: s1 = set(range(3))
      s1
[17]: {0, 1, 2}
[18]: s2 = set("Herman")
      s2
[18]: {'H', 'a', 'e', 'm', 'n', 'r'}
[19]: s1.isdisjoint(s2)
[19]: True
```

```
[17]: s1 = set(range(3))
      s1
[17]: {0, 1, 2}
[21]: s2 = set([1,3])
      s2
[21]: {1, 3}
[22]: s1.isdisjoint(s2)
[22]: False
```

→ because both set contain integer 1

Subset

All items of first set are found in the second

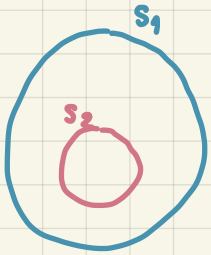
```
[23]: s1 = set(range(10))
      s1
[23]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[24]: s2 = set(range(2,6))
      s2
[24]: {2, 3, 4, 5}
[26]: s2.issubset(s1)
[26]: True
```

compare — compared

Superset

All items of second set are found in the first

```
[27]: s1.issuperset(s2)
[27]: True
```



S_2 is superset for S_1

S_1 is subset for S_2

Intersection

Set contain items that exist in both sets

```
[31]: s1 = set(range(5))
      s1
[31]: {0, 1, 2, 3, 4}
[32]: s2 = set(range(2,9))
      s2
[32]: {2, 3, 4, 5, 6, 7, 8}
[33]: s1.intersection(s2)
[33]: {2, 3, 4}
```

use S_1, S_2

Union

Return a set containing all items from two sets

```
[28]: s1 = set(range(10))
      s1
[28]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
[29]: s2 = set("Herman")
      s2
[29]: {'H', 'a', 'e', 'm', 'n', 'r'}
[30]: s1.union(s2)
[30]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'H', 'a', 'e', 'm', 'n', 'r'}
```

Difference

Items from first set that are not in second

```
[34]: s1.difference(s2)
[34]: {0, 1}
```

→ there are no these numbers in S_2