Built-in Data Types in Python

(set, dictionary)

Massive Information & Knowledge Engineering

204113 Computer & Programming

Dr. Arnon Rungsawang
Dept. of computer engineering
Kasetsart University
https://mike.cpe.ku.ac.th/204113

Version 2024

Python Set

What is a set?

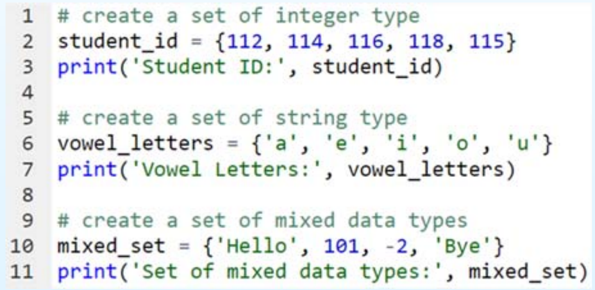
- A set is a collection of unique data.
  - Elements of a set cannot be duplicate.
- Suppose we want to store information about student IDs. Since student IDs cannot be duplicate, we can use a set.

112	114	116	118	115

Set of student IDs

Creating a set

- In Python, we create sets by placing all the elements inside curly braces {}, separated by comma.
- A set can have any number of items and they may be of difference types (integer, float, tuple, string, etc.).
- A set cannot have mutable elements like list, sets of dictionaries as its elements.
- Elements in a set have no particular order.



```
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)

# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)

# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```

Creating an empty set

- Creating an empty set is a bit tricky. Empty curly braces `{}` will make an empty dictionary in Python.
- To make a set without any element, we use the `set()` function without any argument.

```python
# create an empty set
empty_set = set()

# create an empty dictionary
empty_dictionary = { }

# check data type of empty_set
print('Data type of empty_set:', type(empty_set))

# check data type of dictionary_set
print('Data type of empty_dictionary', type(empty_dictionary))
```

Duplicate items in a set

- Let's see what will happen if we try to include duplicate items in a set.

```python
numbers = {2, 4, 6, 6, 2, 8}
print(numbers)    # {8, 2, 4, 6}
```

Add an item to a set

- Sets are mutable. However, since they are unordered, indexing has no meaning.
- We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.
- To add an item to a set in Python, we use the `add()` method.

```python
numbers = {21, 34, 54, 12}

print('Initial Set:',numbers)

# using add() method
numbers.add(32)

print('Updated Set:', numbers)
```

Update items in a set

- The `update()` method is used to update the set with items of other collection types (lists, tuples, sets, etc.).

```python
companies = {'Lacoste', 'Ralph Lauren'}
tech_companies = ['apple', 'google', 'apple']

companies.update(tech_companies)

print(companies)
```

Remove an item from a set

- We use the `discard()` method to remove the specified element from a set.

```python
languages = {'Swift', 'Java', 'Python'}

print('Initial Set:',languages)

# remove 'Java' from a set
languages.discard('Java')

print('Set after remove():', languages)
```

- Note that if the object not found, either do nothing or not raise any exception!

```python
languages = {'Swift', 'Java', 'Python'}

print('Initial Set:',languages)
k = 'Java2'

try:
    if k in languages:
        languages.discard(k)
    else:
        raise Exception(f"Error: {k} not found!")
except Exception as e:
    print(e)

print('Set after remove():', languages)
```

Built-in functions with set

Function	Description
`all()`	Return `True` if all element of the set are true (or if the set is empty).
`any()`	Return `True` if any element of the set are true. If the set is empty, return `False`.
`enumerate()`	Return an enumerate object. It contains the index and value for all the items of the set as a pair.
`len()`	Return the number of items in the set.
`max()`	Return the largest item in the set.
`min()`	Return the smallest item in the set.
`sorted()`	Return a new sorted list from elements in the set (does not sort the set itself).
`sum()`	Return the sum of all elements in the set.

Let try to call the above functions, i.e.,

```
>>> m = {1,False,2}; all(m)
False
```

Iterate over a set

- We can use the `for` loop with set.

```python
fruits = {"Apple", "Peach", "Mango"}

# for loop to access each fruits
for fruit in fruits:
    print(fruit)
```

Set operation (union)

- The union of two sets A and B include all the elements of set A and B.
- We use the `|` operator or the `union()` method to perform the set union operation.

```python
# first set
A = {1, 3, 5}

# second set
B = {0, 2, 4}

# perform union operation using |
m = A | B
print('Union using |:', m)

# perform union operation using union()
m = B.union(A)
print('Union using union():', m)
```

Set operation (intersection)

- The intersection of two sets A and B include common elements between set A and B.
- We use the & operator or the `intersection()` method to perform the set intersection operation.

```python
# first set
A = {1, 3, 5}

# second set
B = {1, 2, 3}

# perform intersection operation using &
m = A & B
print('Intersection using &:', )

# perform ... using intersection()
m = A.intersection(B)
print('.. using intersection():', m)
```

Set operation (difference)

- The difference between two sets A and B include all elements of set A that are not present on set B.
- We use the - operator or the `difference()` method to perform the difference between two set.

```python
# first set
A = {2, 3, 5}

# second set
B = {1, 2, 6}

# perform difference operation using -
m = A - B
print('Difference using -:', m)

# perform ... using difference()
m = A.difference(B)
print('Difference using difference():', m)
```

Set operation (symmetric difference)

- The symmetric difference between two sets A and B include all elements of A and B without the common elements.
- We use the ^ operator or the `symmetric_difference()` method to perform the symmetric difference between two set.

```python
# first set
A = {2, 3, 5}

# second set
B = {1, 2, 6}

# perform difference operation using &
print('using ^:', A ^ B)

# using symmetric_difference()
m = A.symmetric_difference(B)
print('using symmetric_difference():', m)
```

Check if two sets are equal

- We use the == operator to check whether two sets are equal.

```python
# first set
A = {1, 3, 5}

# second set
B = {3, 5, 1}

# perform difference operation using &
if A == B:
    print('Set A and Set B are equal')
else:
    print('Set A and Set B are not equal')
```

Other Python set methods

Method	Description
`add()`	Adds an element to the set
`clear()`	Removes all elements from the set
`copy()`	Return a copy of the set
`difference()`	Returns the difference of two or more sets as a new set
`difference_update()`	Removes all elements of another set from this set
`discard()`	Remove an element from the set if it is a member (Do nothing if the element is not in set)
`intersection()`	Return the intersection of two sets as a new set
`intersection_update()`	Updates the set with the intersection of itself and another
`isdisjoint()`	Return `True` if two sets have a null intersection

Other Python set methods (2)

Method	Description
`issubset()`	Returns `True` if another set contains this set
`issuperset()`	Returns `True` if this set contains another set
`pop()`	Removes and return an arbitrary set element. Raise `KeyError` if the set is empty
`remove()`	Removes an element from the set. If the element is not a member, raise a `KeyError`
`symmetric_difference()`	Return the symmetric difference of two sets as a new set
`symmetric_difference_update()`	Updates a set with the symmetric difference of itself and another
`union()`	Return the union of sets in a new set
`update()`	Updates the set with the intersection of itself and others

Python Dictionary

What's a dictionary

- In Python, a dictionary is a collection that allows us to store data in key-value pairs.
- We create a dictionary by placing `key:value` pairs inside curly brackets `{}`, separated by commas.

```python
# creating a dictionary
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
```

Dictionary key is immutable

- Dictionary keys must be immutable, such as tuples, strings, integers, etc. We cannot use mutable (changeable) objects such as lists as keys.
- We can also use `dict()` function to create dictionaries.

```python
# Valid dictionary
my_dict1 = dict(
  ( (1 , "Hello"),
    [(1, 2) , "Hello Hi"],
    (3 , [1, 2, 3])
  )
)
print(my_dict1)

# Invalid dictionary
# Error: using a list as a key is not allowed
my_dict2 = {
  1: "Hello",
  [1, 2]: "Hello Hi",
}
print(my_dict2)
```

Dictionary length

- We can get the size of a dictionary by using the `len()` function.

```python
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

# get dictionary's length
print(len(country_capitals)) # 3
```

Access dictionary items

- We can access the value of a dictionary item by placing the key inside square brackets [ ].

```python
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

for i in country_capitals.keys():
    print(i, end=', ')
    print(country_capitals[i])

print(country_capitals.keys())
```

Change dictionary items

- Python dictionaries are mutable (changeable).
- We can change the value of a dictionary element by referring to its key.

```python
country_capitals = {
  "United States": "New York",
  "Italy": "Naples",
  "England": "London"
}

# change the value of "Italy" key to "Rome"
country_capitals["Italy"] = "Rome"

print(country_capitals)
```

Add items to a dictionary

- We can add an item to the dictionary by assigning a value to a new key that does not exist in the dictionary.
- We can also use the `update()` method to add or change dictionary items.

```python
country_capitals = {
  "United States": "New York",
  "Italy": "Naples",
  1 : "One",
  '1': "OneStr",
  (1,): "OneTuple"
}

country_capitals["Germany"] = "Berlin"
# note the key parameter in update() method!!!
country_capitals.update(Thailand='Bangkok')
country_capitals.update(Italy='Rome')

#country_capitals.update(1='Un')
#country_capitals.update('1'='UnStr')
#country_capitals.update((1,)='UnTuple')

print(country_capitals)
```

Remove dictionary items

```python
country_capitals = {
  "United States": "New York",
  "Italy": "Naples",
  "Ukrain": "Kiev"
}

del country_capitals["United States"]
print(country_capitals)

co = 'Italy'
ca = country_capitals.pop(co)
print(f'{co}, {ca}')

res = country_capitals.clear()
print(country_capitals, res)
```

- We use `del` statement to remove an element from the dictionary.
- We can also use the `pop()` method to remove an item from the dictionary.
- We can use the `clear()` method to remove all items at once.

Dictionary membership test

- We can check whether a key exists in a dictionary using the `in` operator.
- Note that the `in` operator does not check whether a value exists.

```python
my_list = {1: "Hello", "Hi": 25, "Howdy": 100}

print(1 in my_list) # True

# the not in operator checks whether key doesn't exist
print("Howdy" not in my_list) # False

print("Hello" in my_list) # False
```

Iterating through a dictionary

- A dictionary is an ordered collection of items. Meaning a dictionary maintains the order of its items.
- We can iterate through dictionary keys one by one using a `for` loop.

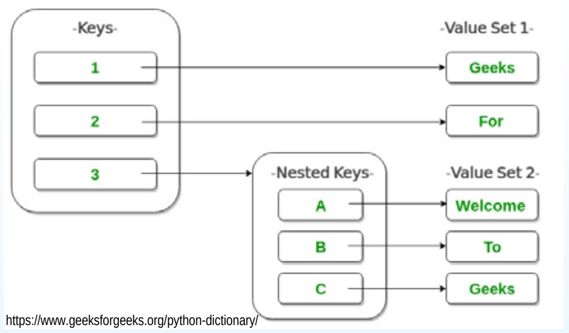
```python
country_capitals = {
  "United States": "New York",
  "Italy": "Naples"
}

# print dictionary keys one by one
for country in country_capitals:
    print(country)

print("----------")

# print dictionary values one by one
for country in country_capitals:
    capital = country_capitals[country]
    print(capital)
```

Nested dictionary



```python
Dict = {1: 'Geeks', 2: 'For',
        3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}

a = Dict[3]['A']
print(a)
Dict[3]['C'] = 'CPE_KU'
print(Dict[3]['C'])
```

Dictionary methods

Method	Description
`pop()`	Remove the item with the specified key
`update()`	Add or change dictionary items
`clear()`	Remove all items from the dictionary
`keys()`	Returns all the dictionary keys
`values()`	Returns all the dictionary values
`get()`	Returns the value of the specified key
`popitem()`	Return the last inserted key and value as a tuple
`copy()`	Returns a copy of the dictionary

Problem Solving Samples

All pair sum to a given value

- Write a program that finds all pairs of elements in a list whose sum is equal to a given value.

```python
res = []
# target = 35
# m = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
target = 7
m = [1,2,3,4,5]
print(f'Input: {m}, target={target}')
for i in range(len(m)):
    complement = target - m[i]
    for j in range(len(m)):
        if j == i:
            continue
        if m[j] == complement:
            res.append([m[i], complement])
            break
print('Output:', res)
```

```
>>> %Run allPairsSumEqGivenValue.py

 Input: [1, 2, 3, 4, 5], target=7
 Output: [[2, 5], [3, 4], [4, 3], [5, 2]]
```

Longest common prefix

- Write a program to find the longest common prefix of all strings.

```python
def lcp(s=["HelloWorld","Hello"]):
    min_length = min([len(word) for word in s])
    for i in range(min_length):
        char_i = set([word[i] for word in s])
        #print(char_i)
        if len(char_i) > 1:
            return s[0][:i]
    #print("#-----------")
    return s[0][:min_length]

s = ["pqrefgh","pqrsfghk"]
#s = ["1234","1234"]
print(f'Input: {s}')
print(f'Output: {lcp(s)}')
```

```
>>> %Run longestCommonPrefix.py

  Input: ['pqrefgh', 'pqrsfghk']
  Output: pqr
```

Max product of all pairs

- Write a program to find the two numbers whose product is maximum among all the pairs of "different numbers" in a given list of numbers.

```python
m = [1, -2, -3, 4, 5, -6, 7, -8, 9, -10]
# m = [1, -2, -3, 4, 5, -6, 7, -8, 9, -5, 5, 9, -10, -10]
print(f'Input: {m}')
maxProd = -99999999
m = list(set(m))
for i in range(len(m)):
    for j in range(i+1, len(m)):
        if m[i]*m[j] > maxProd:
            maxProd = m[i]*m[j]
            a,b = m[i], m[j]
print(a,b)
```

```
>>> %Run maxProd2numbers.py

  Input: [1, -2, -3, 4, 5, -6, 7, -8, 9, -10]
  -10 -8
```

Missing numbers

- Given two sets of numbers, write a program to find the missing numbers in the second set as compared to the first and vice versa.

```python
def missing_numbers1(setN1, setN2):
    res = set(setN1)-set(setN2), set(setN2)-set(setN1)
    return res

def missing_numbers2(setN1, setN2):
    res1, res2 = [], []
    for i in setN1:
        if i not in setN2:
            res1.append(i)
    for j in setN2:
        if j not in setN1:
            res2.append(j)
    return res1, res2

setN1 = {1, 2, 3, 4, 5, 6}
setN2 = {3, 4, 5, 6, 7, 8}
print(f'Input1: {setN1}\nInput2: {setN2}')
a = missing_numbers1(setN1, setN2)
b = missing_numbers2(setN1, setN2)
print(f'{a}\n{b}')
```

```
>>> %Run missingNum.py

  Input1: {1, 2, 3, 4, 5, 6}
  Input2: {3, 4, 5, 6, 7, 8}
  ({1, 2}, {8, 7})
  ([1, 2], [7, 8])
```

Sum of 3 different numbers

- Write a program to find all the unique combinations of 3 "different numbers" from a given list of numbers, adding up to a target number.

```python
m = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target = 17

res = []
for i in range(len(m)):
    for j in range(i+1, len(m)):
        for k in range(j+1, len(m)):
            if m[i]+m[j]+m[k] == target:
                res.append([m[i]]+[m[j]]+[m[k]])
print(f'Input: {m}, target={target}')
print(res)
```

```
>>> %Run combinationOfThree.py

  Input: [1, 2, 3, 4, 5, 6, 7, 8, 9], target=17
  [[1, 7, 9], [2, 6, 9], [2, 7, 8], [3, 5, 9],
  [3, 6, 8], [4, 5, 8], [4, 6, 7]]
```

Frequency of factors

- Given a list with elements, construct a dictionary with frequency of factors.

```python
test_list = [2,6,8,4]
# test_list = [2, 4, 6, 8, 3, 9, 12, 15, 16, 18]
max_elem = max(test_list)
res = {} # create an empty dict
for i in range(1,max_elem+1):
    count = 0
    for k in test_list:
        if k%i==0:
            count += 1
    res[i] = count

print(f'Input: {test_list}')
print(res)
```

```
>>> %Run factorFreqDict.py

 Input: [2, 6, 8, 4]
 {1: 4, 2: 4, 3: 1, 4: 2, 5: 0, 6: 1, 7: 0, 8: 1}
```

Count distinct substrings

```python
'''
Count distinct substrings of a string
'''

s = 'aba'
res = []
print(f'Input: {s}')
for k in range(len(s)):
    for i in range(1,len(s)+1):
        tmp = s[k:i]
        #print(k, i, tmp)
        if tmp != '' and tmp not in res:
            res.append(s[k:i])

print(res)
```

```
>>> %Run countDistinctSubstr.py

 Input: aba
 ['a', 'ab', 'aba', 'b', 'ba']
```

Non-repeating characters

```python
'''
Find all characters of non-repeating characters in a string
'''

s = 'geeksforgeeks'
print(f'Input: {s}')
rep_dic = {}

for c in s:
    if rep_dic.get(c, 'Not Found') == 'Not Found':
        rep_dic[c] = 1
    else:
        rep_dic[c] += 1
print('DEBUG:', rep_dic)
res = []
for k in rep_dic:
    if rep_dic[k] == 1:
        res.append(k)
print('Output:', res)
```

```
>>> %Run findNonRepeatingCharInStr.py

 Input: geeksforgeeks
 DEBUG: {'g': 2, 'e': 4, 'k': 2, 's': 2, 'f': 1, 'o': 1, 'r': 1}
 Output: ['f', 'o', 'r']
```

Word Frequency

- From a collection of written texts, a string of text, also known as corpus, lets create a word frequency with the help of a dictionary.

```python
corpus = '''We learn all about the Pthon Dictionary\
 and its potential. You would also learn to create\
 word frequency using the Dictionary'''

word_freq = dict()
corpus_word = str(corpus).split()
#print(corpus_word)
for i in range(len(corpus_word)):
    if corpus_word[i] not in word_freq:
        word_freq[corpus_word[i]] = 1
    else:
        word_freq[corpus_word[i]] += 1
print(word_freq)
```

```
>>> %Run combinationOfThree.py

 {'We': 1, 'learn': 2, 'all': 1, 'about': 1, 'the': 2, 'Python
 ': 1, 'Dictionary': 2, 'and': 1, 'its': 1, 'potential.': 1, '
 You': 1, 'would': 1, 'also': 1, 'to': 1, 'create': 1, 'word':
 1, 'frequency': 1, 'using': 1}
```

To be continue..

つづく