Python Fundamentals

List, set, dictionary, and string

Minchang (Carson) Zhang

Python Fundamentals

01

LIST

- OPERATIONS
- COMPREHENSION
- ADVANCED OPERATIONS

02

SET

- OPERATIONS

03

DICTIONARY

- OPERATIONS

04

STRING

- FUNDAMENTALS
- ADVANCED

List - operations

- Append
- Extend
- Insert
- Remove
- Pop
- Clear
- Index
- Count
- Sort
- Reverse
- Copy

- a_list.append("element")
- a_list.extend([1,2,3])
- a_list.insert(0, "element")
- a_list.remove("element") # First occurrence
- a_list.pop() or a_list.pop(index) # will return popped element
- a_list.clear() # remove all elements, equivalent to (del a_list[:])
- a_list.index("element") or a_list.index("element", start_index, end_index) # First occurrence
- a_list.count("element") # Return number of occurence
- a_list.sort(*key=None, reverse=False*)
- a_list.reverse()
- a_list.copy()

Guess what happens with list_a = list_b ?

List Comprehension

- Simple comprehension
  - [i for i in a_list]

```
>>> string = "a test sentence string"
>>> a_list = [i for i in string]
>>> a_list
['a', ' ', 't', 'e', 's', 't', ' ', 's', 'e', 'n', 't', 'e',
'n', 'c', 'e', ' ', 's', 't', 'r', 'i', 'n', 'g']
```

- Conditional comprehension
  - [i for i in a_list if i == “element”]

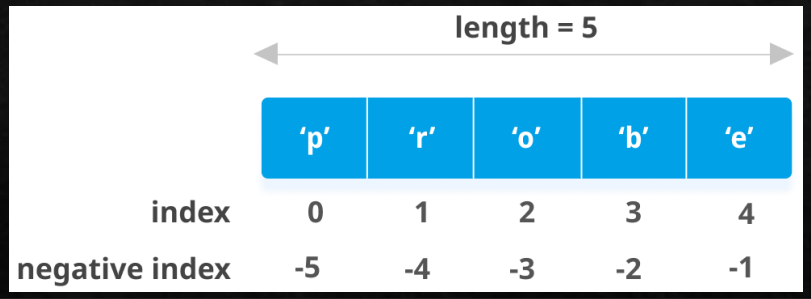
```
>>> string = "a test sentence string"
>>> a_list = [i for i in string if i == "t"]
>>> a_list
['t', 't', 't', 't']
```

- Nested comprehension
  - [i for list1 in list2 for i in list1]

```
>>> a_list = [[1, "a"], [2, "b"], [3, "c"]]
>>> new_list = [i for list1 in a_list for i in list1]
>>> new_list
[1, 'a', 2, 'b', 3, 'c']
```

List – Advanced Operation

- List slicing
  - a_list[:] # all elements
  - a_list[::-1] # Reverse
  - a_list[from_inclusive : to_exclusive : ±step_size]




```
>>> a_list = ["banana", "apple", "peach", "orange", "pear"]
>>> a_list[:]
['banana', 'apple', 'peach', 'orange', 'pear']
>>> a_list[::-1]
['pear', 'orange', 'peach', 'apple', 'banana']
>>> a_list[1:4:2]
['apple', 'orange']
```

- Stack and Queue
  - Stack: first in last out
  - Queue: first in first out
  - DeQueue: Queue with both sides open

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack.pop()
6
>>> stack
[3, 4, 5]
```

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")
>>> queue.append("Graham")
>>> queue.popleft()
'Eric'
>>> queue.popleft()
'John'
>>> queue
deque(['Michael', 'Terry', 'Graham'])
```

List – Advanced Operation

List

```
>>> string = "test"
>>> list(string)
['t', 'e', 's', 't']
>>>
>>> list1 = [1, 2, 3]
>>> list2 = ["Jenny", "Christy", "Monica"]
>>> x = zip(list1, list2)
>>> print(list(x))
[(1, 'Jenny'), (2, 'Christy'), (3, 'Monica')]
```

Enumerate

```
>>> a_list = ["a", "b"]
>>> for index, element in enumerate(a_list):
...     print(f"Element {element} is at index {index}")
...
Element a is at index 0
Element b is at index 1
```

Sort/Sorted

```
>>> a_list = ["banana", "apple", "peach"]
>>> sorted(a_list, key = lambda word: word[1])
['banana', 'peach', 'apple']
```

Sum

```
>>>
>>> a_list = [1, 3, 5, 7]
>>> sum(a_list)
16
>>>
```

Del

```
>>> a_list = ["banana", "apple", "peach"]
>>> del(a_list[1])
>>> a_list
['banana', 'peach']
>>> del(a_list)
>>> a_list
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a_list' is not defined
```

Join

```
>>> a_list = ["banana", "apple", "peach"]
>>> new_string = ", ".join(a_list)
>>> print(f"My fruites are: {new_string}")
My fruites are: banana, apple, peach
```

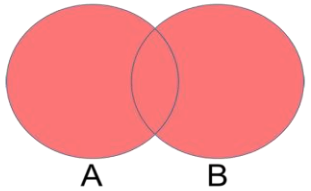
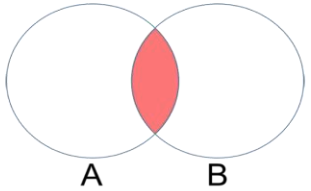
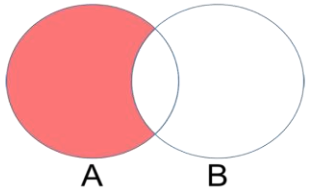
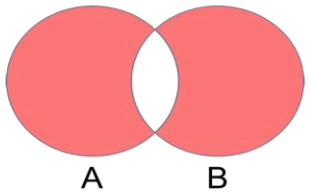
Set - operations

- Definition: set()
- Size: len(set([1,2,3]))
- Modification: .update()
- Addition: .add()
- Deletion: .remove() or .discard() or .pop() or .clear()
- Union: | or .union()
- Intersection: & or .intersection()
- Difference: - or .difference()
- Symmetric Differnece: ^ or .symmetric_difference()
- Is Disjoin: .isdisjoin()
- Is Subset: <= or .issubset()
- Is Superset: >= or .issuperset()

An example:
How to find the common elements between two lists

```
>>> list1 = [1, 3, 5, 2, 6]
>>> list2 = [1, 2, 3, 3, 1, 5, 6, 34, 3]
>>> list(set(list1) & set(list2))
[1, 2, 3, 5, 6]
```

Ref:
Sets in Python
Python sets and Set Theory

Set Operation	Venn Diagram	Interpretation
Union		$A \cup B$, is the set of all values that are a member of A, or B, or both.
Intersection		$A \cap B$, is the set of all values that are members of both A and B.
Difference		$A \setminus B$, is the set of all values of A that are not members of B
Symmetric Difference		$A \triangle B$, is the set of all values which are in one of the sets, but not both.

Dictionary - Operations

- Definition: {key: value} or dict(key=value)
- Access value: dict[key] or .get(key, default=None)
- Clear entire dictionary: .clear()
- Get a list of key-value pairs: .items()
- Get all keys: .keys()
- Get all values: .values()
- Remove a key: .pop(key, default=None) or del(a_dict(key))
- Remove a key-value pair: .popitem() # LIFO
- Update dictionary: .update()

```
>>> a_dict = {'a': 10, 'b': 20, 'c': 30}
>>> a_dict
{'a': 10, 'b': 20, 'c': 30}
>>> a_dict['a']
10
>>> a_dict.get('e', "doesn't exist")
"doesn't exist"
>>> list(a_dict.items())
[('a', 10), ('b', 20), ('c', 30)]
>>> a_dict.keys()
dict_keys(['a', 'b', 'c'])
>>> a_dict.values()
dict_values([10, 20, 30])
>>> a_dict.pop('f', "doesn't exist")
"doesn't exist"
>>> a_dict.popitem()
('c', 30)
>>> a_dict
{'a': 10, 'b': 20}
>>> a_dict.update({'b': 200, 'd': 400})
>>> a_dict
{'a': 10, 'b': 200, 'd': 400}
>>> a_dict.update(b=200, d=400)
>>> a_dict
{'a': 10, 'b': 200, 'd': 400}
>>>
```

String - Fundamentals

- Definition: single quotes, double quotes, triple quotes
- Can be seen as a list, so pretty much all list functions can be used on string
- Formatting: % or .format() or f-string (i.e. f“”)
- Conversion: str()
- Operations:

Option	Meaning
`'<'`	Forces the field to be left-aligned within the available space (this is the default for most objects).
`'>'`	Forces the field to be right-aligned within the available space (this is the default for numbers).
`'='`	Forces the padding to be placed after the sign (if any) but before the digits. This is used for printing fields in the form ‘+000000120’. This alignment option is only valid for numeric types. It becomes the default when ‘0’ immediately precedes the field width.
`'^'`	Forces the field to be centered within the available space.

Option	Meaning
`'+'`	indicates that a sign should be used for both positive as well as negative numbers.
`'-'`	indicates that a sign should be used only for negative numbers (this is the default behavior).
space	indicates that a leading space should be used on positive numbers, and a minus sign on negative numbers.

String - Advanced

- Escape Character

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

- Most Common built-in functions
  - .split()
  - .replace()
  - .strip()
  - .join
  - .capitalize()
  - .upper()
  - .lower()
  - .title()
  - .find()
  - .isalpha()
  - .isdigit()
  - .islower()

```
>>> string = "show/me/the/money     "
>>> string.strip()
'show/me/the/money'
>>> string.replace("o", "11")
'sh11w/me/the/m11ney     '
>>> string.split("/")
['show', 'me', 'the', 'money     ']
>>> new_string = string.strip().replace("/", " ")
>>> new_string
'show me the money'
>>> new_string.capitalize()
'Show me the money'
>>> new_string.upper()
'SHOW ME THE MONEY'
>>> new_string.lower()
'show me the money'
>>> new_string.title()
'Show Me The Money'
>>> new_string.find('e')
6
>>> new_string.islower()
True
>>> new_string.isdigit()
False
```

So we just touched the surface
There are more to discover
Such as str vs repr, regular expression,
built-in functions, dunder methods
Feel free to explore on your own!

Thank you!