Python Datatypes - LIST

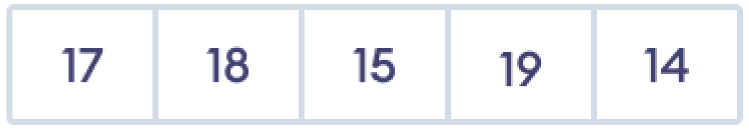
TUYEN NGOC LE

Outline

Python Lists

Python Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In simple language, a list is a collection of things, enclosed in [ ] and separated by commas.

The list is a sequence data type which is used to store the collection of data. Tuples and String are other types of sequence data types.



List of Age

Create a Python List

A list is created in Python by placing items inside **[]**, separated by commas ,

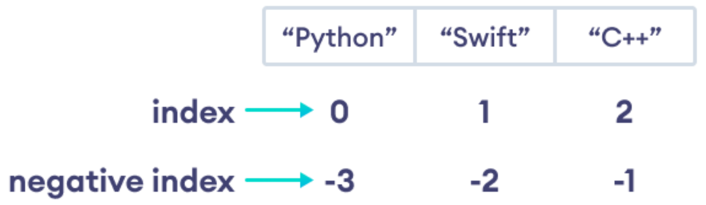
```
# A list with 3 integers
numbers = [1, 2, 5]
print(numbers)
```

A list can have any number of items and they may be of different types (integer, float, string, etc.)

```
# empty list
my_list = []
# list with mixed data types
my_list = [1, "Hello", 3.4]
print(my_list)
```

Access Python List Elements

In Python, each item in a list is associated with a number. The number is known as a list index.



Note: If the specified index does not exist in the list, Python throws the IndexError exception.

```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[0])   # Python
print(languages[-3])   # Python
# access item at index 2
print(languages[2])   # C++
print(languages[-1])   # C++
print(languages[4])
```

Slicing of a Python List

In Python it is possible to access a section of items from the list using the slicing operator : , not just a single item.

```
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# items from index 2 to index 4
print(my_list[2:5])

# items from index 5 to end
print(my_list[5:])

# items beginning to end
print(my_list[:])
```

Iterating through a List

We can use the for loop to iterate over the elements of a list

```
languages = ['Python', 'Swift', 'C++']

# iterating through the list
for item in languages:
    print(item)
```

Python List Length

In Python, we use the len() function to find the number of elements present in a list.

```
languages = ['Python', 'Swift', 'C++']
print("List: ", languages)
print("Total Elements: ", len(languages))
for i in range (len(languages)):
   print (languages[i])
```

Python List Methods

Method	Description
append()	add an item to the end of the list
extend()	add items of lists and other iterables to the end of the list
insert()	inserts an item at the specified index
remove()	removes item present at the given index
pop()	returns and removes item present at the given index
clear()	removes all items from the list
index()	returns the index of the first matched item
count()	returns the count of the specified item in the list
sort()	sort the list in ascending/descending order
reverse()	reverses the item of the list

List append()

The append() method adds an item to the end of the list.
Syntax: list.append(item)

- **item** argument: an item (number, string, list etc.) to be added at the end of the list

```
# animals list
animals = ['cat', 'dog', 'rabbit']

# Add 'guinea pig' to the list
animals.append('guinea pig')

print('Updated animals list: ', animals)
```

```
numbers = []

for x in range(1, 6):
   numbers.append(x * x)
print('Updated numbers list: ', numbers)
```

Python List extend()

extend() method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.

Syntax of List extend(): list. extend(iterable**)**

```
# languages list
languages = ['French']
# another list of language
languages_list = ['Spanish', 'Portuguese']
# languages tuple
languages_tuple = ('Spanish', 'Portuguese')
# languages set
languages_set = {'Chinese', 'Japanese'}
# appending language_list elements to language
languages.extend(languages_list)
print('Languages List:', languages)
# appending language_tuple elements to language
languages.extend(languages_tuple)
print('New Language List:', languages)
# appending language_set elements to language
languages.extend(languages_set)
print('Newer Languages List:', languages)
```

Python extend() Vs append()

```
a1 = [1, 2]
a2 = [1, 2]
b = (3, 4)

# a1 = [1, 2, 3, 4]
a1.extend(b)
print(a1)

# a2 = [1, 2, (3, 4)]
a2.append(b)
print(a2)
```

Python List insert()

insert() method inserts an element to the list at the specified index.

Syntax of List insert(): list.insert(i, item)

Here, item is inserted to the list at the i^th index. All the elements after item are shifted to the right.

```
mixed_list = [{1, 2}, [5, 6, 7]]
# number tuple
number_tuple = (3, 4)
# inserting a tuple to the list
mixed_list.insert(1, number_tuple)
print('Updated List:', mixed_list)
```

Python List remove()

remove() method removes the first matching element (which is passed as an argument) from the list.

```
# animals list
animals = ['cat', 'dog', 'dog', 'guinea pig', 'dog']
# 'dog' is removed
animals.remove('dog')
# Updated animals list
print('Updated animals list: ', animals)
# Deleting 'fish' element
animals.remove('fish')
```

```
ValueError: list.remove(x): x not in list
```

```
# animals list
animals = ['cat', 'dog', 'dog', 'guinea pig', 'dog']
# 'dog' is removed
animals.remove('dog')

# Updated animals list
print('Updated animals list: ', animals)
# Deleting 'fish' element
if 'fish' in animals:
   animals.remove('fish')
```

Python List count()

method returns the number of times the specified element appears in the list.

Syntax of List count(): list.count(element)

- **element** - the element to be counted

Return value from count(): the number of times element appears in the list

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
# count element 'i'
count = vowels.count('i')
# print count
print('The count of i is:', count)
# count element 'p'
count = vowels.count('t')
# print count
print('The count of t is:', count)
```

Python List pop()

pop() method removes the item at the given index from the list and returns the removed item.

Syntax of List pop(): list.pop(index)

pop() parameters:
The pop() method takes a single argument (index).
The argument passed to the method is optional. If not passed, the default index -1 is passed as an argument (index of the last item).
If the index passed to the method is not in range, it throws IndexError: pop index out of range exception.

Return Value from pop(): returns the item present at the given index. This item is also removed from the list.

Python List pop()

```
# programming languages list
languages = ['Python', 'Java', 'C++', 'French', 'C']
# remove and return the 4th item
return_value = languages.pop(3)
print('Return Value:', return_value)
# Updated List
print('Updated List:', languages)
```

pop() without an index, and for negative indices

```
# programming languages list
languages = ['Python', 'Java', 'C++', 'Ruby', 'C']

# remove and return the last item
print('When index is not passed:')
print('Return Value:', languages.pop())
print('Updated List:', languages)
# remove and return the last item
print('\nWhen -1 is passed:')
print('Return Value:', languages.pop(-1))
print('Updated List:', languages)
# remove and return the third last item
print('\nWhen -3 is passed:')
print('Return Value:', languages.pop(-3))
print('Updated List:', languages)
```

If you need to remove the given item from the list, you can use the remove() method. And, you can use the del statement to remove an item or slices from the list.

Python List reverse()

reverse() method reverses the elements of the list.

Syntax of List reverse(): list.reverse()

```
#Operating System List
systems = ['Windows', 'macOS', 'Linux']
print('Original List:', systems)
# List Reverse
systems.reverse()
# updated list
print('Updated List:', systems)
for o in reversed(systems):
    print(o)
```

#Reverse a List Using Slicing Operator

```
# Operating System List
systems = ['Windows', 'macOS', 'Linux']
print('Original List:', systems)
# Reversing a list
# Syntax: reversed_list = systems[start:stop:step]
reversed_list = systems[::-1]
# updated list
print('Updated List:', reversed_list)
for o in reversed(systems):
    print(o)
```

Python List sort()

sort() method sorts the items of a list in ascending or descending order.

sort() Syntax: list.sort(key=..., reverse=...)
reverse - If True, the sorted list is reversed (or sorted in Descending order)
key - function that serves as a key for the sort comparison

sort() Return Value:
The sort() method doesn't return any value. Rather, it changes the original list.
If you want a function to return the sorted list rather than change the original list, use sorted().

```
# Sort in Descending order
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']
# sort the vowels
vowels.sort()
# print vowels
print('Sorted list:', vowels)
```

```
# Sort the list in Descending order
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']
# sort the vowels
vowels.sort(reverse=True)
# print vowels
print('Sorted list (in Descending):', vowels)
```

Sort with custom function using key

If you want your own implementation for sorting, the sort() method also accepts a key function as an optional parameter.

```
# Sort with custom function using key
# sorting using custom key
employees = [
    {'Name': 'Alan Turing', 'age': 25, 'salary': 10000},
    {'Name': 'Sharon Lin', 'age': 30, 'salary': 8000},
    {'Name': 'John Hopkins', 'age': 18, 'salary': 1000},
    {'Name': 'Mikhail Tal', 'age': 40, 'salary': 15000},
]
# custom functions to get employee info
def get_name(employee):
    return employee.get('Name')
def get_age(employee):
    return employee.get('age')
def get_salary(employee):
    return employee.get('salary')
# sort by name (Ascending order)
employees.sort(key=get_name)
print(employees, end='\n\n')
# sort by Age (Ascending order)
employees.sort(key=get_age)
print(employees, end='\n\n')
# sort by salary (Descending order)
employees.sort(key=get_salary, reverse=True)
print(employees, end='\n\n')
```

Sort with custom function using key

```
# sorting using custom key
employees = [
   {'Name': 'Alan Turing', 'age': 25, 'salary': 10000},
   {'Name': 'Sharon Lin', 'age': 30, 'salary': 8000},
   {'Name': 'John Hopkins', 'age': 18, 'salary': 1000},
   {'Name': 'Mikhail Tal', 'age': 40, 'salary': 15000},
]
# sort by name (Ascending order)
employees.sort(key=lambda x: x.get('Name'))
print(employees, end='\n\n')
# sort by Age (Ascending order)
employees.sort(key=lambda x: x.get('age'))
print(employees, end='\n\n')
# sort by salary (Descending order)
employees.sort(key=lambda x: x.get('salary'), reverse=True)
print(employees, end='\n\n')
```

built-in sorted() function

The sorted() function sorts the elements of a given iterable in a specific order (ascending or descending) and returns it as a list.

Note: The simplest difference between sort() and sorted() is: sort() changes the list directly and doesn't return any value, while sorted() doesn't change the list and returns the sorted list.
If you want a function to return the sorted list rather than change the original list, use sorted().

Syntax of sorted(): sorted(iterable, key=None, reverse=False)

sorted() Parameters:
iterable - A sequence (string, tuple, list) or collection (set, dictionary, frozen set) or any other iterator.
reverse (Optional) - If True, the sorted list is reversed (or sorted in descending order). Defaults to False if not provided.
key (Optional) - A function that serves as a key for the sort comparison. Defaults to None.

built-in sorted() function

Sort string, list, and tuple

```
# vowels list
py_list = ['e', 'a', 'u', 'o', 'i']
print(sorted(py_list))

# string
py_string = 'Python'
print(sorted(py_string))

# vowels tuple
py_tuple = ('e', 'a', 'u', 'o', 'i')
print(sorted(py_tuple))
```

built-in sorted() function

Sort in descending order

```
# set
py_set = {'e', 'a', 'u', 'o', 'i'}
print(sorted(py_set, reverse=True))

# dictionary
py_dict = {'e': 1, 'a': 2, 'u': 3, 'o': 4, 'i': 5}
print(sorted(py_dict, reverse=True))

# frozen set
frozen_set = frozenset(('e', 'a', 'u', 'o', 'i'))
print(sorted(frozen_set, reverse=True))
```

built-in sorted() function

Sort the list using sorted() having a key function

```
# take the second element for sort
def take_second(elem):
    return elem[1]

# random list
random = [(2, 2), (3, 4), (4, 1), (1, 3)]

# sort list with key
sorted_list = sorted(random, key=take_second)

# print list
print('Sorted list:', sorted_list)
```

built-in sorted() function

Sorting with multiple keys

```
# Nested list of student's info in a Science Olympiad
# List elements: (Student's Name, Marks out of 100 , Age)
participant_list = [
    ('Alison', 50, 18),
    ('Terence', 75, 12),
    ('David', 75, 20),
    ('Jimmy', 90, 22),
    ('John', 45, 12)
]
def sorter(item):
    # Since highest marks first, least error = most marks
    error = 100 - item[1]
    age = item[2]
    return (error, age)
sorted_list = sorted(participant_list, key=sorter)
print(sorted_list)
```

built-in sorted() function

using the lambda function

```
# Nested list of student's info in a Science Olympiad
# List elements: (Student's Name, Marks out of 100 , Age)
participant_list = [
    ('Alison', 50, 18),
    ('Terence', 75, 12),
    ('David', 75, 20),
    ('Jimmy', 90, 22),
    ('John', 45, 12)
]
sorted_list = sorted(participant_list, key=lambda item: (100-
item[1], item[2]))
print(sorted_list)
```

Python List copy()

The copy() method returns a shallow copy of the list. The copy() method returns a new list. It doesn't modify the original list.

Syntax: new_list = list.copy()

```
# mixed list
old_list = ['cat', 0, 6.7]
# copying a list
new_list = my_list.copy()
new_list.append('a')
print('Copied List:', new_list)
print('Old List:', old_list)
```

```
# List copy using “=“
old_list = [1, 2, 3]
# copy list using =
new_list = old_list
# add an element to list
new_list.append('a')
print('New List:', new_list)
print('Old List:', old_list)
```

Python List copy()

Copy List Using Slicing Syntax

```
# shallow copy using the slicing syntax
# mixed list
list = ['cat', 0, 6.7, 'cat', 1, 1.2]
# copying a list using slicing
new_list = list[1:2]

# Adding an element to the new list
new_list.append('dog')

# Printing new and old list
print('Old List:', list)
print('New List:', new_list)
```

Python List clear()

clear() method removes all items from the list. The clear() method only empties the given list. It doesn't return any value.

Syntax: list.clear()

```
# Defining a list
list = [{1, 2}, ('a'), ['1.1', '2.2']]
# clearing the list
list.clear()
print('List:', list)
```

```
# Emptying the List Using del
# Defining a list
list = [{1, 2}, ('a'), ['1.1', '2.2']]
# clearing the list
del list[:]
print('List:', list)
```

Exercises

1. Write a Python program to sum all the items in a list.

2. Write a Python program to multiply all the items in a list

3. Write a Python program to get the largest, smallest number from a list

4. Write a Python program to remove duplicates from a list.

5. Write a Python program to find the list of words that are longer than n from a given list of words.

6. Write a Python program to find items starting with a specific character from a list.
Original list: ['abcd', 'abc', 'bcd', 'bkie', 'cder', 'cdsw', 'sdfsd', 'dagfa', 'acjd’]
Expected Output:
+ Items start with a from the said list: ['abcd', 'abc', 'acjd’]
+ Items start with d from the said list: ['dagfa’]
+ Items start with w from the said list: []

References

- https://realpython.com/what-can-i-do-with-python/
- https://www.w3schools.com/python/python_getstarted.asp
- https://www.programiz.com/python-programming/first-program
- https://www.geeksforgeeks.org/
- https://www.w3resource.com/python-exercises/list/

Thank you for listening!