Data Structures

Bachelor's Degree in Electrical and Mechanical Engineering

Carlos III University of Madrid

uc3m | Universidad Carlos III de Madrid
Departamento de Informática

Types of Data Structures

1. **List** is an ordered and modifiable collection. Allows duplicate elements
2. **Tuple** is an ordered and immutable collection. Allows duplicate elements
3. **Set** is a messy collection and not indexed. There are no duplicate elements.
4. **String** is a collection of ordered and modifiable characters

URLs:

- https://www.w3schools.com/python/python_lists.asp
- https://docs.python.org/3.6/tutorial/introduction.html

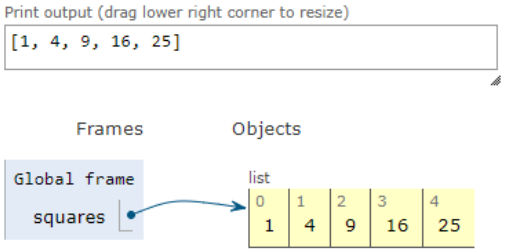
List (I)


- DAILY ACTIVITY -
CLEANING HOUSE
EAT HEALTHY FOOD
WORKOUT
FEEDING PET
DIRTY CLOTHES
designed by freepik.com

List (II)

- The list of special variables that store several elements
- It can be written as a list of values separated by commas (items) in brackets
- It is not necessary that the items in a list all have the same type even if it is generally preferable that they are of the same type.
- The first element of the list is in position 0

```python
squares = [1, 4, 9, 16, 25]
print(squares)
```



List (III)

```
cars = ["Ford", "Volvo", "BMW"]
```

VS

```
car1 = "Ford";
car2 = "Volvo";
car3 = "BMW";
```

List (IV)

Get the value of the first element in the list:

```
x = cars [0] # the first item in the list is in position 0
```

Modify the first item in the list:

```
cars [0] = "Toyota"
```

Know the number of items in the "cars" list:

```
x = len (cars)
```

List (V)

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

thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

List and Loop

Print the elements in the list “cars”:

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

Methods (I)

Method	Description
append(element)	Adds an element to the **end** of the list
clear()	Deletes all elements of the list
copy()	Returns a copy of the list
count(item)	Returns the number of elements with the specified value (item)
extend(list)	Adds the elements of another list to the end of the current list
index(item)	Returns the index of the first element with the specified value (item)
insert(pos, item)	Adds an element (item) at the specified position (pos)
pop([pos])	Retrieves and deletes an element from the list at the specified position (pos) or the element at the end.
remove(item)	Removes the first element with the specified value
reverse()	Reverses (inverts) the order of the list
sort()	Orders the list

Methods (II)

- fruits = ['apple', 'banana', 'cherry', 'orange']
  fruits.append("orange")

- fruits = ['apple', 'banana', 'cherry', 'orange']
  fruits.clear()

- fruits = ['apple', 'banana', 'cherry', 'orange']
  x = fruits.copy()

- fruits = ['apple', 'banana', 'cherry']
  x = fruits.count("cherry")

Methods (III)

- ```
  fruits = ['apple', 'banana', 'cherry']
  cars = ['Ford', 'BMW', 'Volvo']
  fruits.extend(cars)
  ```

- ```
  fruits = ['apple', 'banana', 'cherry']
  x = fruits.index("cherry")
  ```

- ```
  fruits = ['apple', 'banana', 'cherry']
  fruits.insert(1, "orange")
  ```
  ```

Methods (IV)

- fruits = ['apple', 'banana', 'cherry']
  fruits.remove("banana")

- fruits = ['apple', 'banana', 'cherry']
  fruits.reverse()

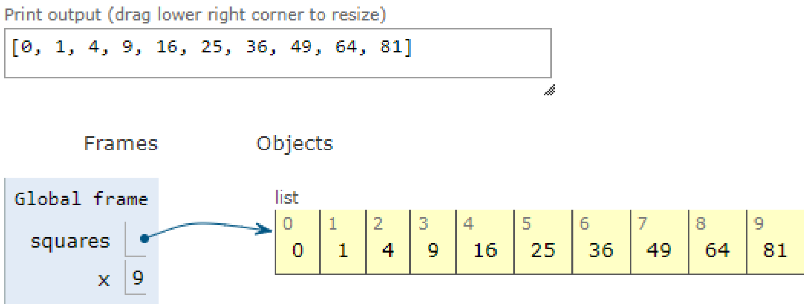
- cars = ['Ford', 'BMW', 'Volvo']
  cars.sort()

- fruits = ['apple', 'banana', 'cherry']
  fruits.pop(1)

Examples

Fill a list with a succession of squares of 10 elements

```python
squares = []

for x in range(10):
    squares.append(x * x)

print(squares)

```



Examples (more)

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4)  # Find next banana starting a position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
>>> fruits.pop()
'pear'
```

The lower in a list

```
vec = [2,3,5,9,1,-1,2,3]

low = vec[0] # need to start with some value
for i in vec:
    if i < low:
        low = i
print (low)
```

Order a list

```
list=[23, 123,1, 5, 0]
for index in range(1,len(list)):
    value = list[index]
    i = index-1
    while i>=0:
        if value < list[i]:
            list[i+1] = list[i]
            list[i] = value
            i -= 1
        else:
            break
print (list)
```

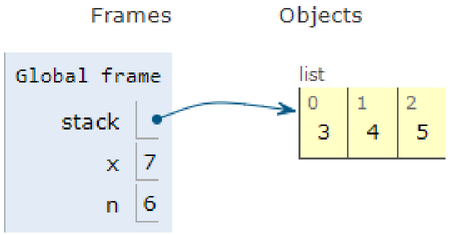
Stack (LIFO)

- To use a list as a stack it is only allowed to use functions *append* and *pop*.

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

print("Original Stack:", stack)

x = stack.pop()
print("Last element:", x)
print("Modified Stack:", stack)

n = stack.pop()
print("Last element:", n)
print("Modified Stack:", stack)
```

Print output (drag lower right corner to resize)

```
Original Stack: [3, 4, 5, 6, 7]
Last element: 7
Modified Stack: [3, 4, 5, 6]
Last element: 6
Modified Stack: [3, 4, 5]
```



Queue (FIFO)

- The most eficciency way to create a queue is by using the class *deque* from the module *collections*.

```
from collections import deque
myQueue = deque([3,4,5])
```

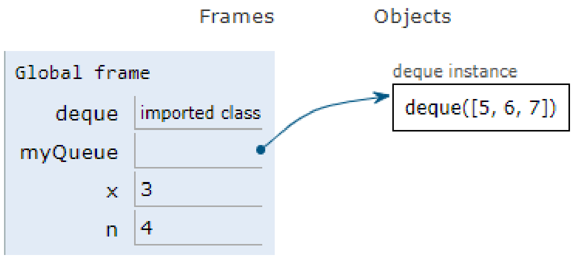
- The deque class contains the *append* and *popleft* functions to use the structure as a queue.

Queue (FIFO)

```python
from collections import deque
myQueue = deque([3,4,5])

myQueue.append(6)
myQueue.append(7)

print("Original queue:", myQueue)

x = myQueue.popleft()
print("First element:", x)
print("Modified Stack:", myQueue)

n = myQueue.popleft()
print("First element:", n)
print("Modified Stack:", myQueue)
```

Print output (drag lower right corner to resize)

```
Original queue: deque([3, 4, 5, 6, 7])
First element: 3
Modified Stack: deque([4, 5, 6, 7])
First element: 4
Modified Stack: deque([5, 6, 7])
```



Tuple

Tuple

- A tuple is an immutable list. **It can not be modified in any case after its creation.**

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
print(len(thistuple))
> banana
> 3
```

```
thistuple = ("apple", "banana", "cherry")
thistuple[1] = "blackcurrant" # Forbidden!!!
```

Set

Set

- A Set is a collection with no order and not indexed. There are no duplicate elements

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
> {'apple', 'cherry', 'banana '}

thisset.add("damson")
thisset.remove("banana")
print(len(thisset))
> 3
```

Set

- You can use ‘remove’ or ‘discard’ to remove elements from a set
  - If the element doesn’t exist, remove will raise an error
  - If the element doesn’t exist, discard will **not** raise an error

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
> {'apple', ' cherry', 'banana '}

thisset.add("damson")
thisset.remove("banana")
thisset.discard("orange")
```

String

Chain of characters

String (I)

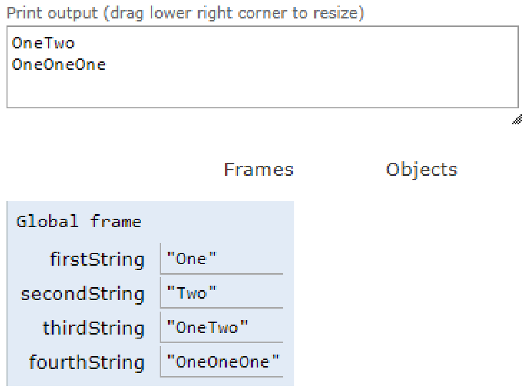
- Chains are nothing more than text enclosed in single quotes ('string') or double quotes (“string”).
- Within the quotes you can add special characters by escaping them with '\', such as '\n', the new line character, or '\t', the tab character.

`'hello'` is the same as `"hello"`.

String (II)

- It is also possible to enclose a string between triple quotes (single or double). In this way we can write the text in several lines, and when printing the string, the line breaks that we introduced will be respected
- Chains also support operators such as addition (chain concatenation) and multiplication.

```python
firstString = "One"
secondString = "Two"

thirdString = firstString + secondString
print(thirdString)

fourthString = firstString * 3
print(fourthString)
```



String (III)

- Chains can be printed on the screen using the **print** function.

- A **character is a string of length 1**.

- As in a list the **brackets allow access to the character**.

String (IV)

Get the character at position 1:

```
a = "hello"
print(a[1])
```

Get the characters from position 2 to 5:

```
b = "world"
print(b[2:5])
```

String (V)

The **strip** method returns the string without leading and trailing spaces:

```
a = "      Hello, World!    "
print(a.strip()) # returns "Hello, World!"
```

Function **len**() returns the length of the String:

```
a = "Hello, World!"
print(len(a))
```

String (VI)

The **lower**() method returns the string in lowercase:

```
a = "Hello, World!"
print(a.lower())
> hello, world!
```

The **upper**() method returns the string in uppercase:

```
a = "Hello, World!"
print(a.upper())
> HELLO, WORLD!
```

String (VII)

replace() replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
> Jello, World!
```

split() divides the string into substrings when and if it finds the separator:

```
a = "Hello, World!"
print(a.split(","))
> ['Hello', ' World!']
```