



fit@hcmus

VNUHCM - UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

University of Science - VNU-HCM
Faculty of Information Science
Department of Computer Science

MTH083 - Advanced Programming for Artificial Intelligence

Slot 03- Function & List

Advisor:

Dr. Nguyễn Tiến Huy

Dr. Lê Thanh Tùng

- 1 Function
- 2 List
- 3 Sorting Algorithms

Part 1: Function

- A function is a block of code that performs a specific task
- Dividing a complex problem into smaller chunks makes our program easy to understand and reuse

Python Function Declaration

The syntax to declare a function is:

```
def function_name(arguments):  
    # function body  
  
    return
```

Here,

- `def` - keyword used to declare a function
- `function_name` - any name given to the function
- `arguments` - any value passed to function
- `return` (optional) - returns value from a function

- First, declare and define a function

```
def greet():  
    print('Hello World!')
```

- However, it does not run until we call it

```
# call the function  
greet()
```

- Think about data types of parameters and return value

```
def function_name(parameter: data_type) -> return_type:  
    """Doctring"""  
    # body of the function  
    return expression
```

```
def add(num1: int, num2: int) -> int:  
    """Add two numbers"""  
    num3 = num1 + num2  
  
    return num3
```

- Information can be passed into functions as arguments
- All parameters (arguments) in the Python language are passed by reference
- Types of arguments:
- Default argument
- Variable-length arguments

- A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument
- Once we have a default argument, all the arguments to its right must also have default values

```
def myFun(x, y=50):  
    print("x: ", x)  
    print("y: ", y)
```

```
# Driver code (We call myFun() with only  
# argument)  
myFun(10)
```


- Write a function to check whether an integer is prime or not?

- Write a function to find the smallest prime number which is bigger than the positive integer k

- In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:
- `*args` (Non-Keyword Arguments) – tuple of arguments
- `**kwargs` (Keyword Arguments) – a dictionary of arguments

```
def myFun(*argv):  
    for arg in argv:  
        print(arg)
```

```
def myFun(**kwargs):  
    for key, value in kwargs.items():  
        print("%s == %s" % (key, value))  
  
# Driver code  
myFun(first='Geeks', mid='for', last='Geeks')
```

```
myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')
```

- The first string after the function is called the Document string or Docstring in short
- This is used to describe the functionality of the function

```
def evenOdd(x):  
    """Function to check if the number is even or odd"""  
  
    if (x % 2 == 0):  
        print("even")  
    else:  
        print("odd")  
  
# Driver code to call the function  
print(evenOdd.__doc__)
```

- In Python every variable name is a reference
- When we pass a variable to a function, a new reference to the object is created

```
# Here x is a new reference to same list lst
```

```
def myFun(x):  
    x[0] = 20
```

Output

```
[20, 11, 12, 13, 14, 15]
```

```
# Driver Code (Note that lst is modified  
# after function call.  
lst = [10, 11, 12, 13, 14, 15]  
myFun(lst)  
print(lst)
```

- When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken

```
def myFun(x):
```

```
    # After below line link of x with previous  
    # object gets broken. A new object is assigned  
    # to x.  
    x = [20, 30, 40]
```

```
# Driver Code (Note that lst is not modified  
# after function call.  
lst = [10, 11, 12, 13, 14, 15]  
myFun(lst)  
print(lst)
```

Output

```
[10, 11, 12, 13, 14, 15]
```

Pass by Reference or pass by value **fit@hcmus**

- When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken

```
def myFun(x):
```

```
    # After below line link of x with previous  
    # object gets broken. A new object is assigned  
    # to x.  
    x = 20
```

```
# Driver Code (Note that lst is not modified  
# after function call.  
x = 10  
myFun(x)  
print(x)
```

Output

10

- Guest the result of the following code:

```
def swap(x, y):  
    temp = x  
    x = y  
    y = temp
```

```
# Driver code  
x = 2  
y = 3  
swap(x, y)  
print(x)  
print(y)
```


- Guest the result of the following code:

```
def swap(x, y):  
    temp = x  
    x = y  
    y = temp
```

```
# Driver code  
x = 2  
y = 3  
swap(x, y)  
print(x)  
print(y)
```

Output

2

3

- Write a function to find the maximum value in a list of integer

- Write a function to count the prime number in a list of integer

- Write a function to sort a list of integer via selection sort

- Write a function to determine whether a list of integer is symmetrical or not ?

- Write a function to simplify a fraction
 - Find GCD

Part 2: List

- List:
 - is used to store multiple items in a single variable
 - is just like dynamically sized arrays
 - is a collection of things
 - List items are ordered, changeable/mutable, and allow duplicate values

- Declare a list: A list is created in Python by placing items inside [], separated by commas

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

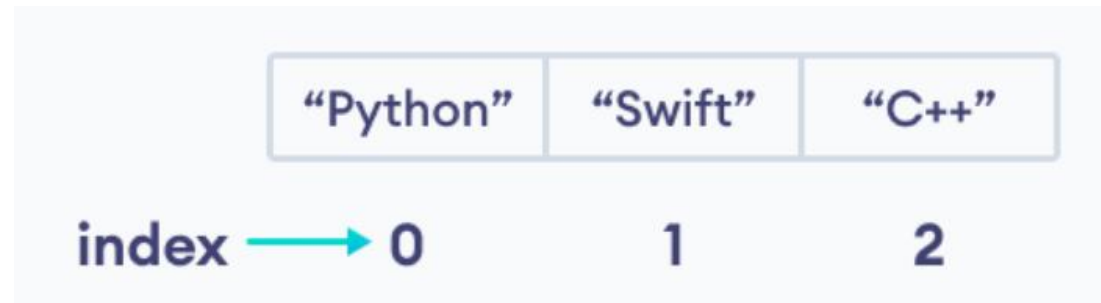
print(numbers)

# Output: [1, 2, 5]
```

- Access an item in list
 - Via item index

```
languages = ["Python", "Swift", "C++"]  
  
# access item at index 0  
print(languages[0])    # Python  
  
# access item at index 2  
print(languages[2])    # C++
```

- Access an item in list
 - Via item index



```
languages = ["Python", "Swift", "C++"]  
  
# access item at index 0  
print(languages[0])    # Python  
  
# access item at index 2  
print(languages[2])    # C++
```

- Python allows negative indexing for its sequences.
- The negative index is in reverse way

	"Python"	"Swift"	"C++"
index →	0	1	2
negative index →	-3	-2	-1

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

# access item at index 0
print(languages[-1])    # C++

# access item at index 2
print(languages[-3])    # Python
```

- A way to access a section of items from the list
- Slicing operator: 

```
# 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[:])
```

- General syntax: `Lst[Initial : End : IndexJump]`
- returns the portion of the list from index *Initial* to index *End*, at a step size *IndexJump*
- Default:
 - *IndexJump* = 1
 - *Initial* = 0
 - *End* = *len(Lst)*

```
# Initialize list
Lst = [50, 70, 30, 20, 90, 10, 50]
```

```
# Display list
print(Lst[::])
```

```
[50, 70, 30, 20, 90, 10, 50]
```

- What is the output of the following code:

```
1  # Initialize list
2  Lst = [50, 70, 30, 20, 90, 10, 50]
3
4  # Display list
5  print(Lst[::-1])
```

- What is the output of the following code:

```
1  # Initialize list
2  Lst = [50, 70, 30, 20, 90, 10, 50]
3
4  # Display list
5  print(Lst[::-1])
```

```
[50, 10, 90, 20, 30, 70, 50]
```

- If some slicing expressions are made that do not make sense or are incomputable, then **empty lists** are generated

- What is the output of the following code:

```
1  # Initialize list
2  Lst = [50, 70, 30, 20, 90, 10, 50]
3
4  # Display list
5  print(Lst[::-1])
```

```
[50, 10, 90, 20, 30, 70, 50]
```

- `len()`

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

- Use the pre-defined functions:
 - Append
 - Insert
 - Extend
 - List concatenation: by operator +

```
list.append(elmnt)
```

- Add **element** in the end of list

Parameter Values

Parameter	Description
<i>elmnt</i>	Required. An element of any type (string, number, object etc.)

```
a = ["apple", "banana", "cherry"]
```

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

```
a.append(b)
```

```
['apple', 'banana', 'cherry', ["Ford", "BMW", "Volvo"]]
```

The syntax of the `insert()` method is

```
list.insert(i, elem)
```

Here, `elem` is inserted to the list at the `ith` index. All the elements after `elem` are shifted to the right.

```
# create a list of prime numbers
prime_numbers = [2, 3, 5, 7]

# insert 11 at index 4
prime_numbers.insert(4, 11)

print('List:', prime_numbers)
```

```
List: [2, 3, 5, 7, 11]
```

Extend

Syntax of List extend()

iterable:

list

tuple

string

The syntax of the `extend()` method is:

```
list1.extend(iterable)
```

Here, all the elements of `iterable` are added to the end of `list1`.

```
# languages list
languages = ['French', 'English']

# another list of language
languages1 = ['Spanish', 'Portuguese']

# appending language1 elements to language
languages.extend(languages1)

print('Languages List:', languages)
```

```
Languages List: ['French', 'English', 'Spanish', 'Portuguese']
```

- we can concatenate two lists by operators +
- `a + b` means `a.extend(b)`

```
a = [1, 2]
b = [3, 4]

a += b    # a = a + b

# Output: [1, 2, 3, 4]
print('a =', a)
```

- If two continuous items are both odd, add their sum between them.
 - For example, if we have the list `[1, 3, 5, 3]`,
 - the output is: `[1, 4, 3, 8, 5, 8, 3]`

- Pre-defined functions:
 - `clear()`
 - `pop()`
 - `remove()`
 - `del`

Syntax of List clear()

The syntax of `clear()` method is:

```
list.clear()
```

Return Value from clear()

The `clear()` method only empties the given [list](#). It doesn't return any value.

```
# Defining a list
list = [{1, 2}, ('a'), ['1.1', '2.2']]

# clearing the list
list.clear()

print('List:', list)
```

remove

Syntax of List remove()

The syntax of the `remove()` method is:

```
list.remove(element)
```

remove() Parameters

- The `remove()` method takes a single element as an argument and removes it from the list.
- If the `element` doesn't exist, it throws **ValueError: list.remove(x): x not in list** exception.

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

# 'rabbit' is removed
animals.remove('rabbit')

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

```
Updated animals list:  ['cat', 'dog', 'guinea pig']
```

```
# animals list
animals = ['cat', 'dog', 'dog', 'guinea pig', 'dog']

# 'dog' is removed
animals.remove('dog')

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

```
Updated animals list:  ['cat', 'dog', 'guinea pig', 'dog']
```

- If a list contains duplicate elements, the **remove()** method only removes the **first** matching element

Syntax of List pop()

The syntax of the `pop()` method is:

```
list.pop(index)
```

- Default: index = -1

Return Value from pop()

The `pop()` method returns the item present at the given index. This item is also removed from the list.

```
# 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)
```

```
Return Value: French
Updated List: ['Python', 'Java', 'C++', 'C']
```

The Python `del` keyword is used to delete objects. Its syntax is:

```
# delete obj_name  
del obj_name
```

Here, `obj_name` can be variables, user-defined objects, lists, items within lists, dictionaries etc.

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
# deleting the third item  
del my_list[2]  
  
# Output: [1, 2, 4, 5, 6, 7, 8, 9]  
print(my_list)  
  
# deleting items from 2nd to 4th  
del my_list[1:4]
```

```
# Output: [1, 6, 7, 8, 9]  
print(my_list)  
  
# deleting all elements  
del my_list[:]  
  
# Output: []  
print(my_list)
```

- `index()`
- `count()`
- `len()`
- `sort()`
- `reverse()`
- `copy()`

index

list index() parameters

The list `index()` method can take a maximum of three arguments:

- **element** - the element to be searched
- **start** (optional) - start searching from this index
- **end** (optional) - search the element up to this index

Return Value from List index()

- The `index()` method returns the index of the given element in the list.
- If the element is not found, a `ValueError` exception is raised.

Note: The `index()` method only returns the first occurrence of the matching element.

Syntax of List index()

The syntax of the list `index()` method is:

```
list.index(element, start, end)
```

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']

# index of 'e' in vowels
index = vowels.index('e')
print('The index of e:', index)

# element 'i' is searched
# index of the first 'i' is returned
index = vowels.index('i')

print('The index of i:', index)
```

- Write a program to return the index of all occurrences of the matching element in list

A = [1, 2, 2, 3, 2, 2]

Element = 2

Return [1, 2, 4, 5]

count()

Syntax of List count()

The syntax of the `count()` method is:

```
list.count(element)
```

count() Parameters

The `count()` method takes a single argument:

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

Return value from count()

The `count()` method returns 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('p')

# print count
print('The count of p is:', count)
```

Output

```
The count of i is: 2
The count of p is: 0
```

- Write a program to count all prime numbers in a list of positive integers

Syntax

Following is the syntax for **len()** method –

```
len(list)
```

Parameters

- **list** – This is a list for which number of elements to be counted.

Return Value

This method returns the number of elements in the list.

```
List.sort(reverse=True|False, key=myFunc)
```

Parameter Values

Parameter	Description
reverse	Optional. reverse=True will sort the list descending. Default is reverse=False
key	Optional. A function to specify the sorting criteria(s)

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

```
cars.sort(reverse=True)
```

```
print(cars)
```

```
['Volvo', 'Ford', 'BMW']
```

Example

Sort the list by the length of the values:

```
# A function that returns the length of the value:
def myFunc(e):
    return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(key=myFunc)
```

```
['VW', 'BMW', 'Ford', 'Mitsubishi']
```

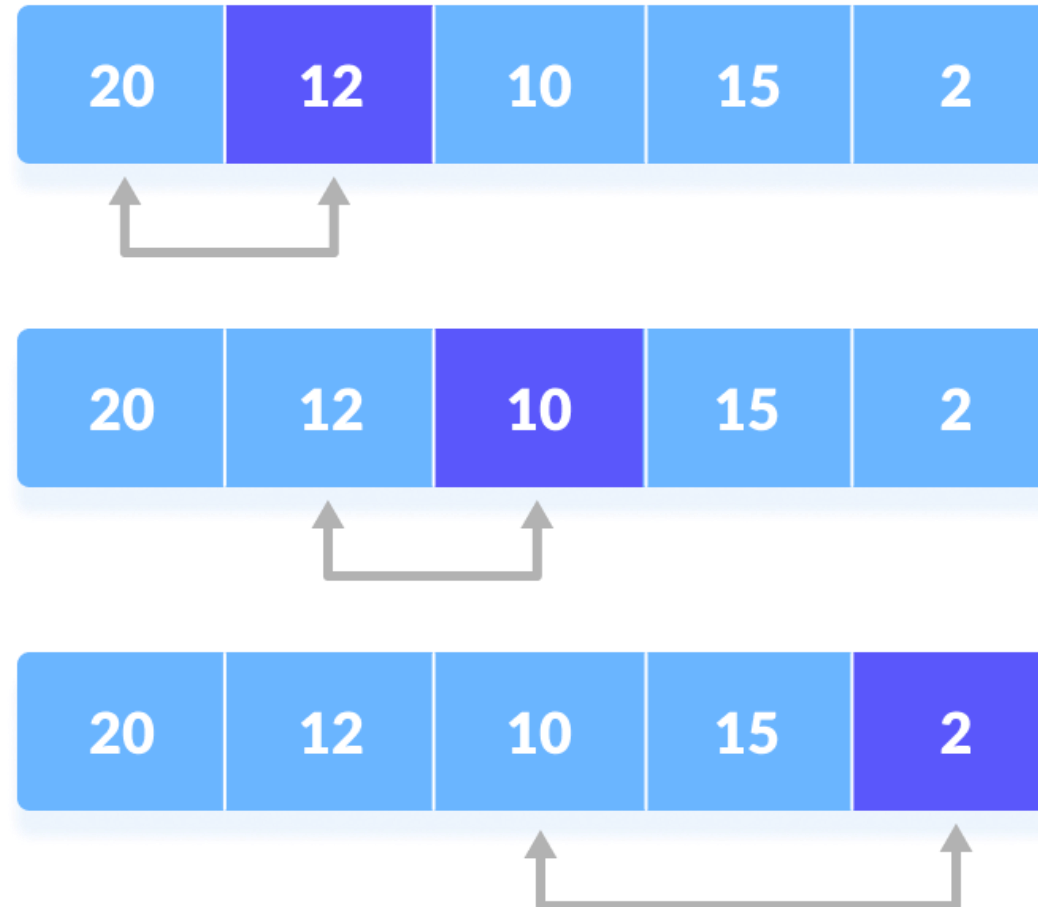
Part 3: Sorting Algorithms

- In this course, we only introduce
 - selection sort
 - bubble sort

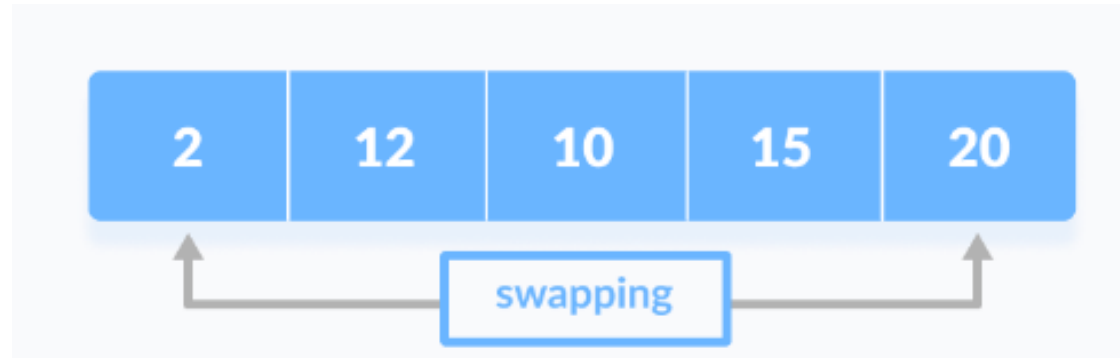
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(\log(n))$
Mergesort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
Timsort	$O(n)$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
Heapsort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Shell Sort	$O(n)$	$O((n \log(n))^2)$	$O((n \log(n))^2)$	$O(1)$
Bucket Sort	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$

- Main idea: repeatedly doing the following procedure:
- finding the minimum element (considering ascending order) from unsorted part
- putting it at the beginning

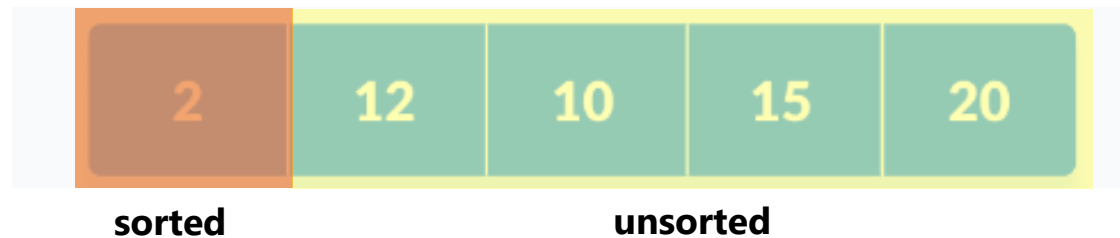
- Step 1: find the minimum value of the list



- Step 2: min val is placed in the front of the unsorted list



- Step 3: repeatedly step 1-2 for the unsorted parts

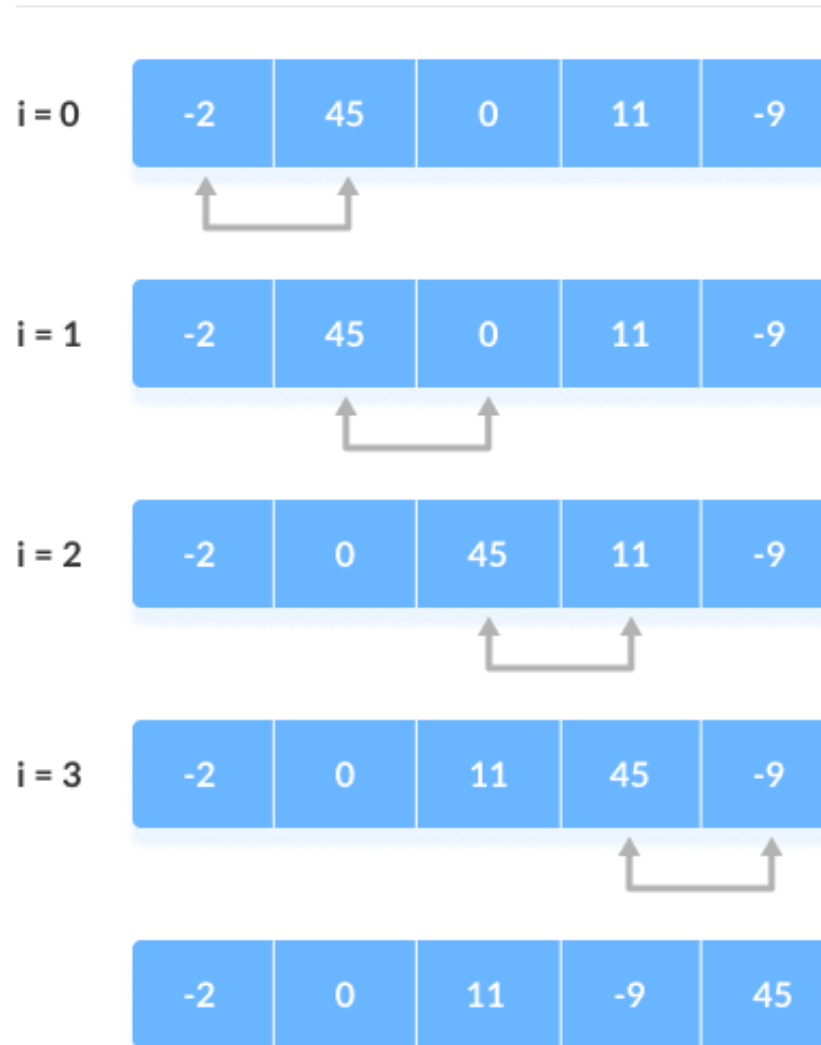


```
def selectionSort(array, size):  
  
    for step in range(size):  
        min_idx = step  
  
        for i in range(step + 1, size):  
  
            # to sort in descending order, change > to < in this line  
            # select the minimum element in each loop  
            if array[i] < array[min_idx]:  
                min_idx = i  
  
        # put min at the correct position  
        (array[step], array[min_idx]) = (array[min_idx], array[step])  
  
data = [-2, 45, 0, 11, -9]  
size = len(data)  
selectionSort(data, size)  
print('Sorted Array in Ascending Order:')  
print(data)
```

- Main idea: compares two adjacent elements and swaps them until they are in the intended order

- Step 1: Compare and Swap
- If i^{th} and $(i+1)^{\text{th}}$ elements are in the incorrect positions, swap them

step = 0



- Step 2: Remaining Iteration
- Repeat Step 1
- Until sorted list or $\text{len}(\text{list}) - 1$ times

```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
  end bubbleSort
```



```
# Bubble sort in Python

def bubbleSort(array):

    # loop to access each array element
    for i in range(len(array)):

        # loop to compare array elements
        for j in range(0, len(array) - i - 1):

            # compare two adjacent elements
            # change > to < to sort in descending order
            if array[j] > array[j + 1]:

                # swapping elements if elements
                # are not in the intended order
                temp = array[j]
                array[j] = array[j+1]
                array[j+1] = temp

data = [-2, 45, 0, 11, -9]

bubbleSort(data)

print('Sorted Array in Ascending Order:')
print(data)
```

1. Write a program to move all the odd numbers into the left-hand side and the even numbers into the right-hand side. Note that the relative positions of all elements are remained and not using the temporary list (in-place algorithm)

e.g:

the unsorted list: [4, 5, 1, 7, 9, 3, 0, 2]

the sorted list: [5, 1, 7, 9, 3, 4, 0, 2]

THANK YOU
for YOUR ATTENTION