

EXERCISE: Data Structure 2



AI VIET NAM
[@aivietnam.edu.vn](http://aivietnam.edu.vn)



- **Review: Python Data Structures**
 - Primitive và Non-primitive Data Structures
- **Exercise1: Viết function dùng dictionary data đếm số chữ và từ**
 - Dictionary đếm số chữ trong 1 từ
 - Dictionary đếm số từ trong các câu của 1 file
- **Exercise2: Xây dựng stack data type bằng dictionary và list**
 - Stack với các method cơ bản: CreateStack, IsEmptyStack, IsFullStack, PopStack, PushStack, GetTopStack
- **Exercise3: Xây dựng queue data type bằng dictionary và list**
 - Queue với các method cơ bản: CreateQueue, IsEmptyQueue, IsFullQueue, Dequeue, Enqueue, GetFirstValue



- **Exercise4: Application of Stack - Stock Span Problem**
 - Tính số ngày tối đa liên tiếp từ ngày hiện tại cho đến các ngày trước trong quá khứ có giá cổ phiếu thấp hơn hoặc bằng giá cổ phiếu hiện tại
- **Exercise5: Application of Queue - Generate Binary number from 1 to N**
 - Tạo ra chuỗi binary từ 1 đến N và đưa kết quả vào một dictionary
- **Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA**
 - Tìm và trả về một list các chuỗi có 10 ký tự được lặp lại hơn một lần trong chuỗi DNA cho trước

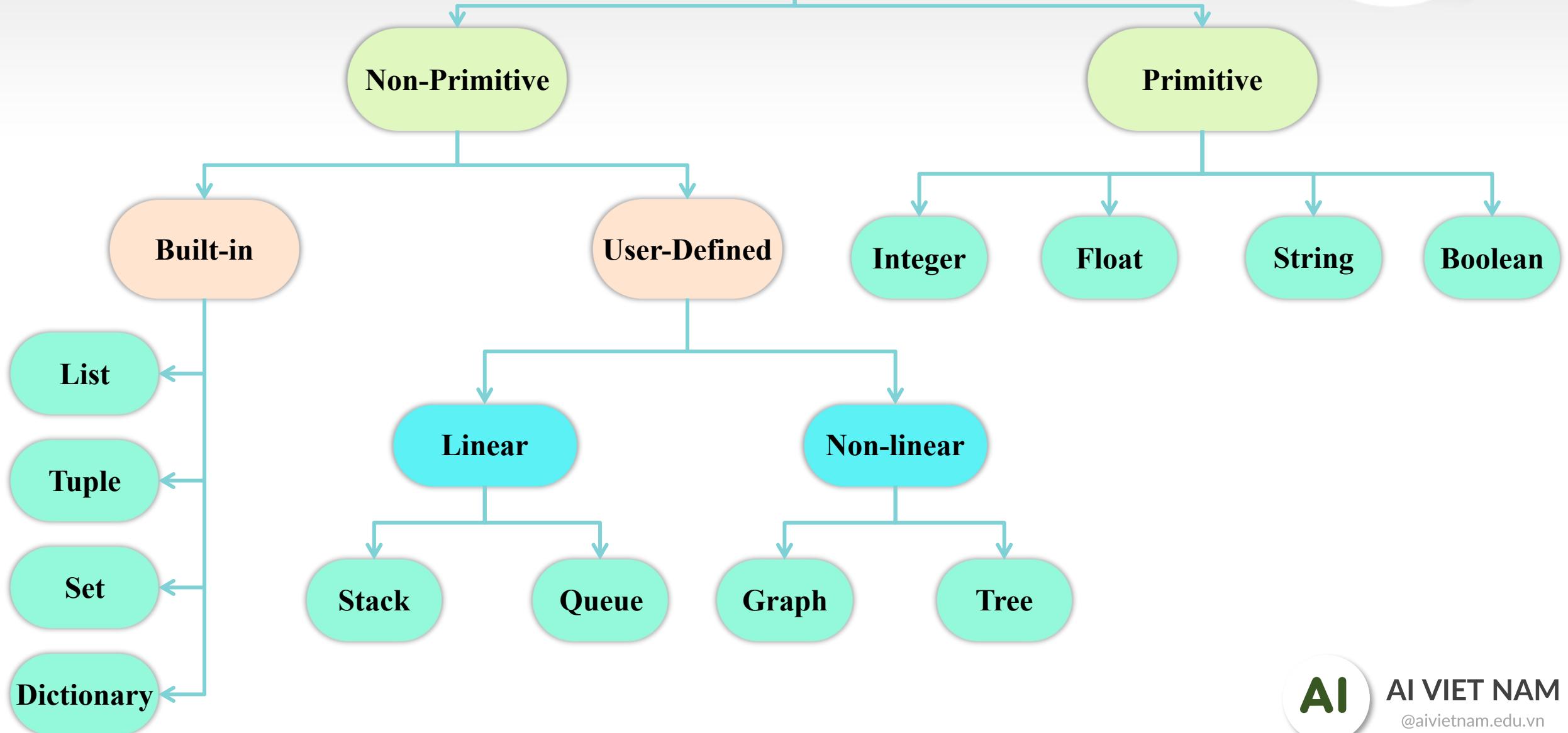


- **Review: Python Data Structures**
 - Primitive và Non-primitive Data Structures
- **Exercise1: Viết function dùng dictionary data đếm số chữ và từ**
 - Dictionary đếm số chữ trong 1 từ
 - Dictionary đếm số từ trong các câu của 1 file
- **Exercise2: Xây dựng stack data type bằng dictionary và list**
 - Stack với các method cơ bản: CreateStack, IsEmptyStack, IsFullStack, PopStack, PushStack, GetTopStack
- **Exercise3: Xây dựng queue data type bằng dictionary và list**
 - Queue với các method cơ bản: CreateQueue, IsEmptyQueue, IsFullQueue, Dequeue, Enqueue, GetFirstValue

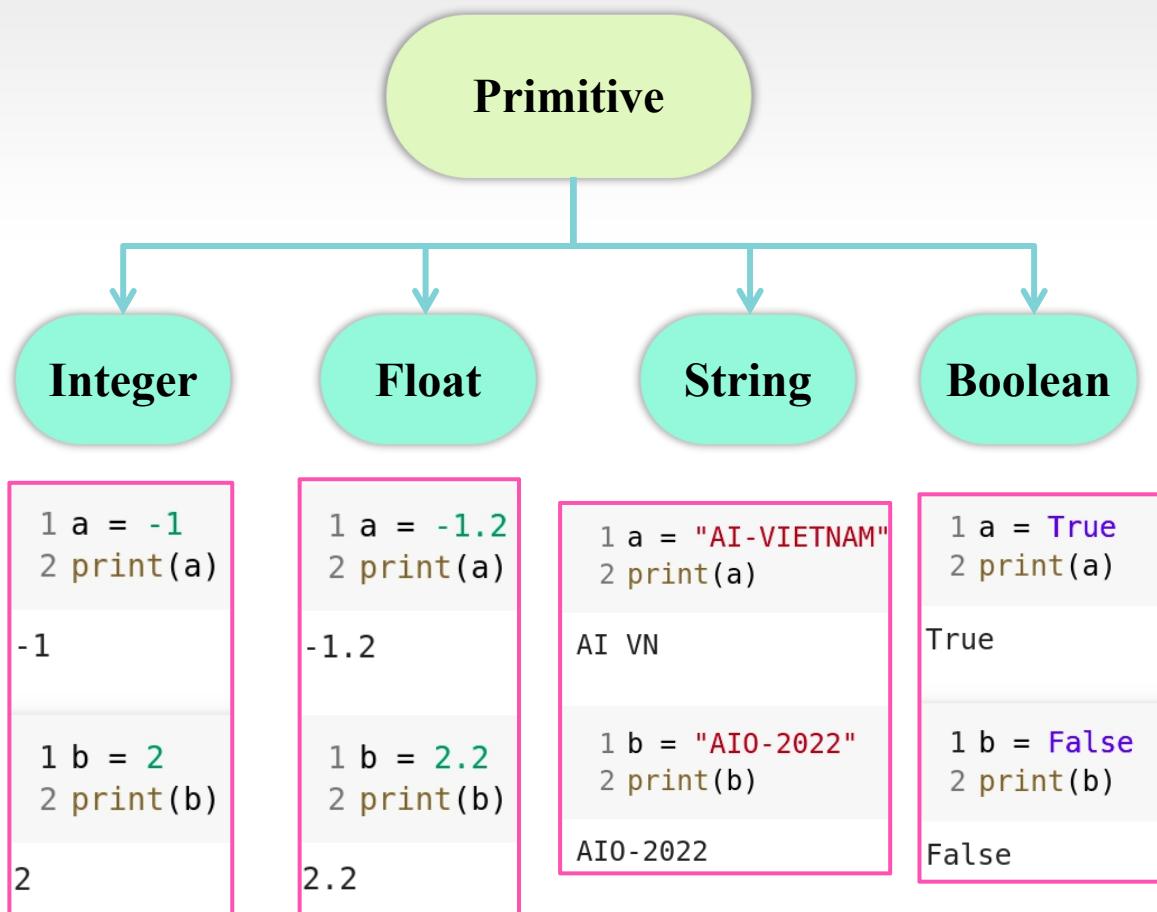
Review: Python Data Structures



Data Structure

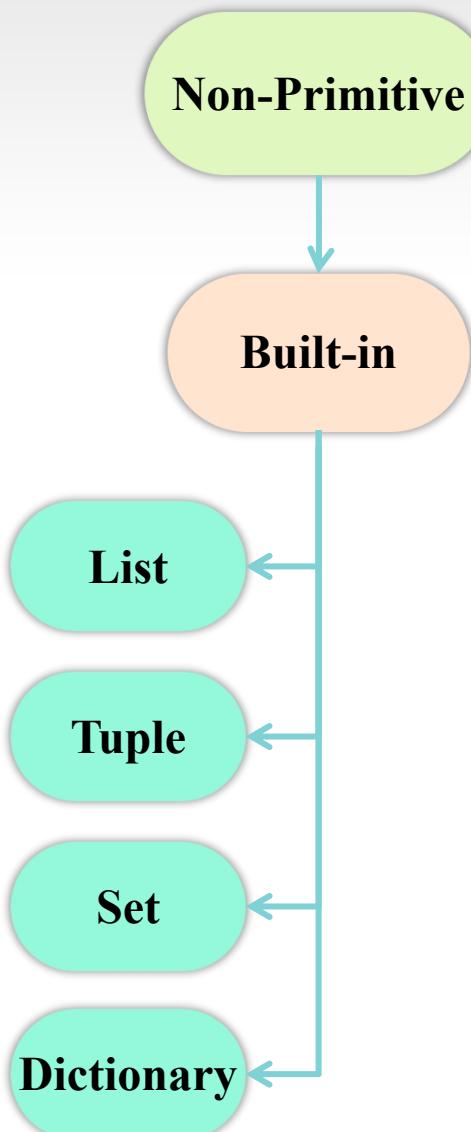


Review: Python Data Structures



Primitive Data Structures

- Basic data structures
- Contain pure, simple values of a data
- **Integer**: -1, 2, 3, ...
- **Float**: 10.1, 1.5, ...
- **String**: “AI VIETNAM”, ...
- **Boolean**: True or False



Non-Data Structures

- Pre-installed with Python
- Making programming easier
- Allowing programmers to get solutions faster
- A collection of values in various formats
- **List**: [“AI VIETNAM”, 2022, 15.15]
- **Tuple**: 10.1, 1.5, ...
- **String**: “AI VIETNAM”, ...
- **Boolean**: True or False

Review: Python Data Structures

Non-Primitive

```
1 a = ["AI VIETNAM", 2022, 15.15]
2 print(a)
['AI VIETNAM', 2022, 15.15]
```

List:

- Indexed by integers starting from zero
- Negative indexing starting from -1
- Store collection of heterogeneous items
- Mutable

Built-in

```
1 a = ("AI VIETNAM", 2022, 15.15)
2 print(a)
('AI VIETNAM', 2022, 15.15)
```

Tuple:

- Indexed by integers starting from zero
- Negative indexing starting from -1
- Store collection of heterogeneous items
- Immutable

List

```
1 a = {15.15, "AI VIETNAM", 2022, 1, 15.15}
2 print(a)
{1, 'AI VIETNAM', 2022, 15.15}
```

Set:

- An unordered collection
- No duplicate elements
- Items are unchangeable, but you can remove items and add new items

Tuple

Set

```
1 thisdict = {
2     "name": "AI VIETNAM",
3     "year": 2022,
4     "ID": 10
5 }
6 print(thisdict["name"])
AI VIETNAM
```

Dictionary:

- Key-Value pairs
- Key is immutable and unique, Val is mutable
- Python version 3.7+: ordered

AI

AI VIET NAM
@aivietnam.edu.vn

Review: Python Data Structures



Non-Primitive

User-Defined

Linear

Stack

Queue

Stack

TOP

10

12

31

Stack:

- **Pre-defined capacity**, can store the elements of a limited size.
- An **ordered list** (**order** in which the **data arrives** is **important**)
- **Insertion** and **Deletion** are done **at one end** called **top**.
- Top pointer (**top**) pointing to the topmost element of the stack
- The **last element inserted** is the **first one to be deleted**. Last in First out (**LIFO**) principle

Stack Operations:

- **Push(value)**: insert data onto stack
- **Pop()**: remove and return the last inserted element from the stack

Auxiliary stack operations:

- **Top()**: return the last inserted element (top) without removing it
- **IsEmptyStack()**: indicate whether the stack is empty or not
- **IsFullStack()**: indicate whether the stack is full or not

Stack Exceptions:

- **Overflow**: Try to push an element to a full stack
- **Underflow**: Try to pop out an empty stack

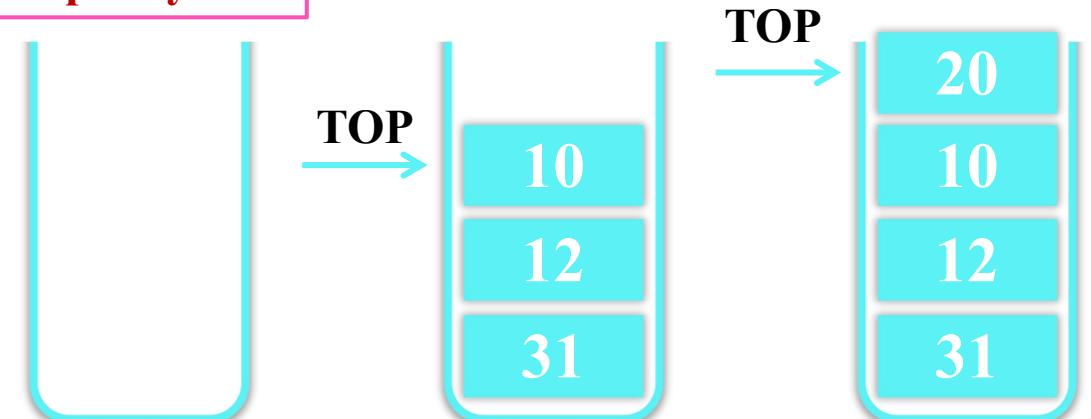
AI

AI VIET NAM
@aivietnam.edu.vn

Review: Python Data Structures



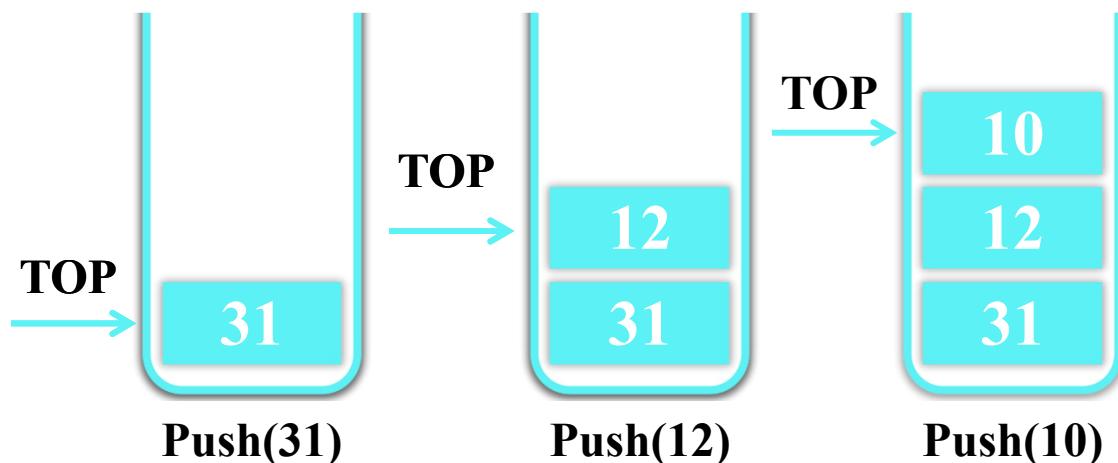
Capacity = 4



Empty Stack

Top() = 10

Full Stack

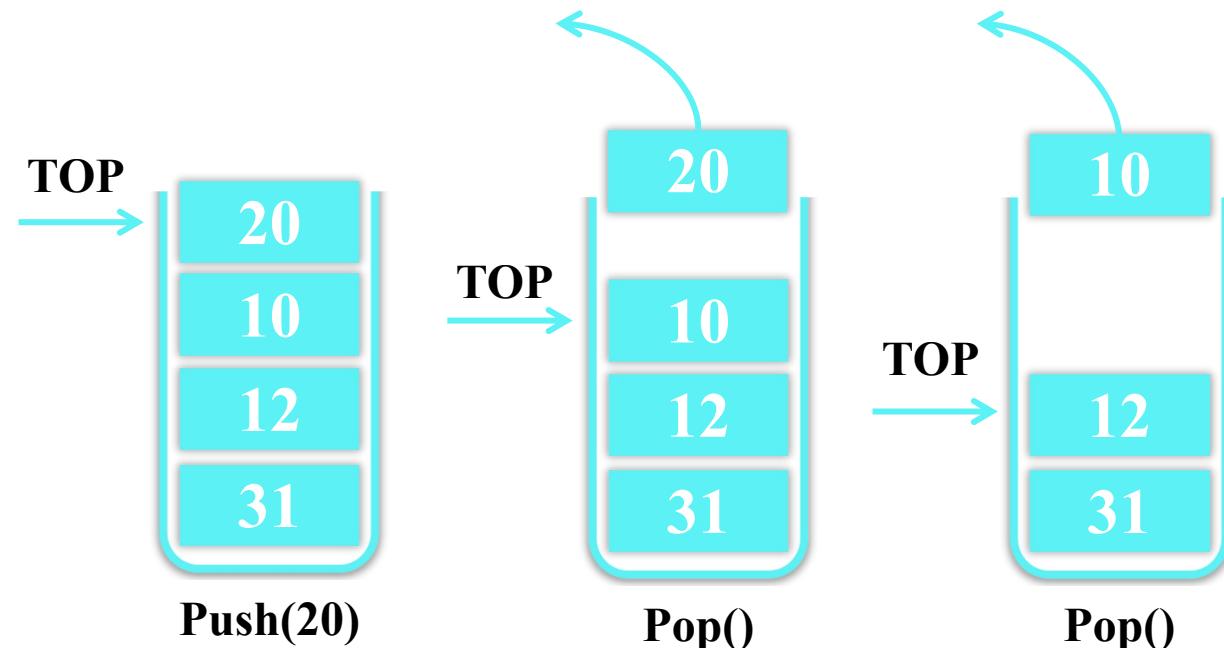


Stack Operations:

- Push(value): insert data onto stack
- Pop(): remove and return the last inserted element from the stack

Auxiliary stack operations:

- Top(): return the last inserted element (top) without removing it
- IsEmptyStack(): indicate whether the stack is empty or not
- IsFullStack(): indicate whether the stack is full or not

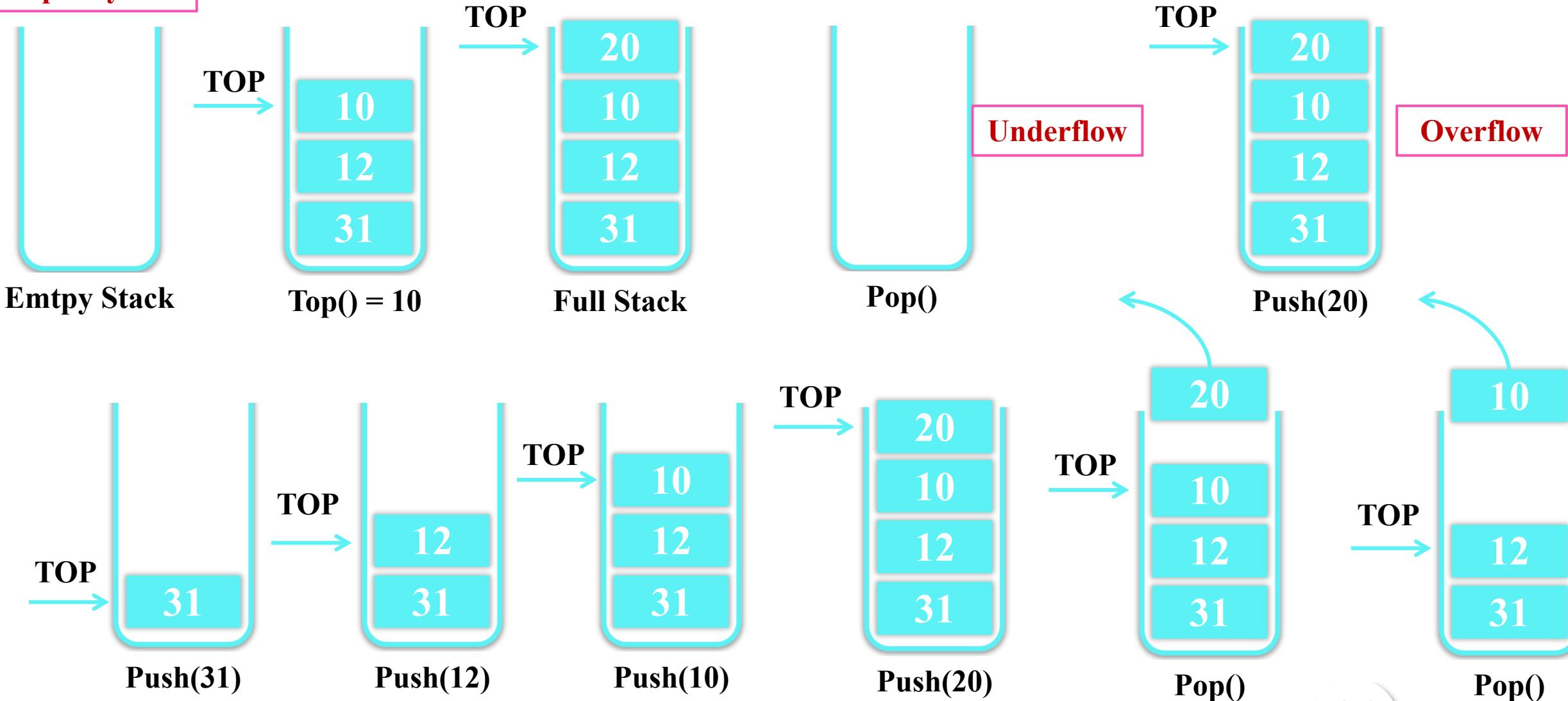


Review: Python Data Structures

Stack Exceptions:

- **Overflow:** Try to push an element to a full stack
- **Underflow:** Try to pop out an empty stack

Capacity = 4



Review: Python Data Structures



Non-Primitive



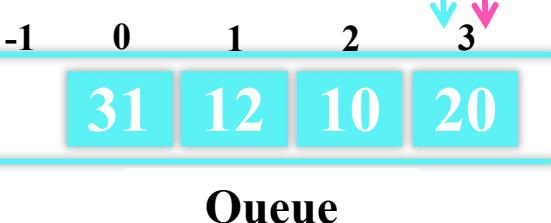
User-Defined



Linear

Stack

Queue



Queue:

- **Pre-defined capacity**, can store the elements of a limited size.
- An **ordered list** (**order** in which the **data arrives** is **important**)
- **Insertions** are done at one end (**rear**)
- **Deletions** are done at other end (**front**)
- Rear pointer (**rear**) pointing to the **last** element of the queue
- Front pointer (**front**) pointing to the **first** element of the queue
- The **first element inserted** is the **first one to be deleted**. First in First out (**FIFO**) principle

Queue Operations:

- **Enqueue(value)**: insert data onto queue
- **Dequeue()**: remove and return the first inserted element from the queue

Auxiliary stack operations:

- **Front()**: return the first inserted element (front) without removing it
- **IsEmptyQueue()**: indicate whether the queue is empty or not
- **IsFullQueue()**: indicate whether the queue is full or not

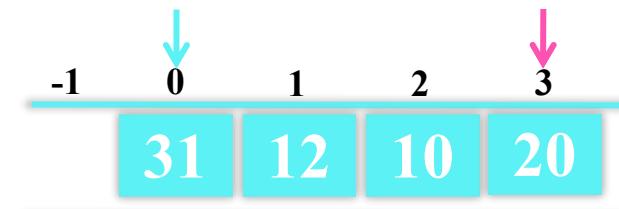
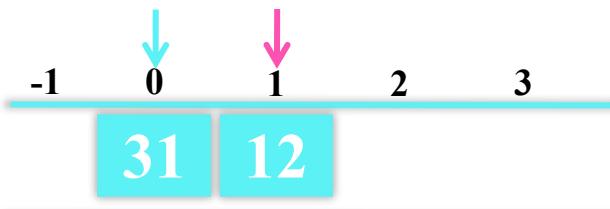
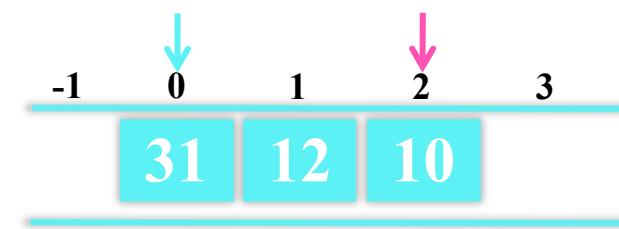
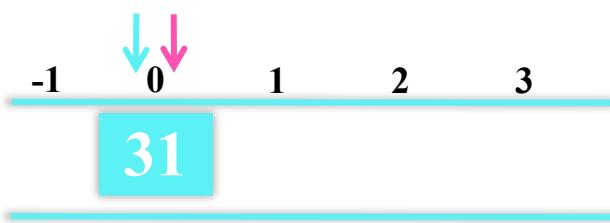
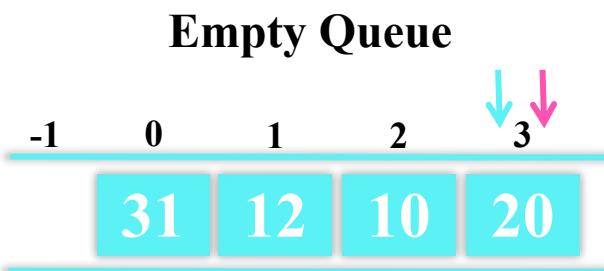
Queue Exceptions:

- **Overflow**: Try to enqueue an element to a full queue
- **Underflow**: Try to dequeue an empty queue

Review: Python Data Structures



Capacity = 4

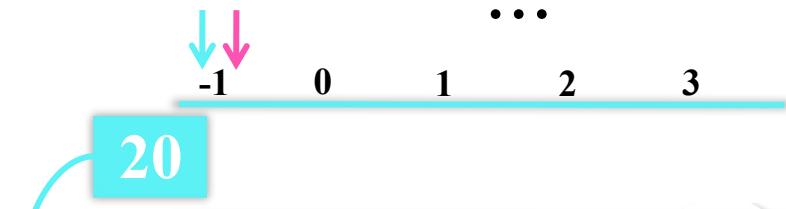
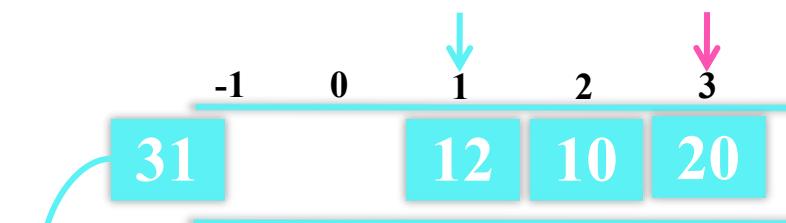


Queue Operations:

- **Enqueue(value):** insert data onto queue
- **Dequeue():** remove and return the first inserted element from the queue

Auxiliary stack operations:

- **Front():** return the first inserted element (front) without removing it
- **IsEmptyQueue():** indicate whether the queue is empty or not
- **IsFullQueue():** indicate whether the queue is full or not



Review: Python Data Structures

Queue Exceptions:

- **Overflow:** Try to enqueue an element to a full queue
- **Underflow:** Try to dequeue an empty queue

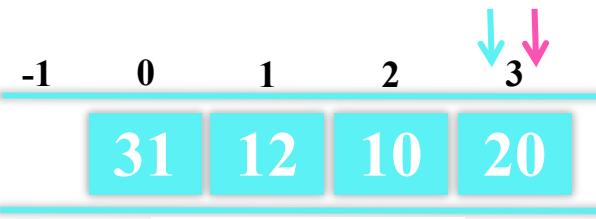
Capacity = 4



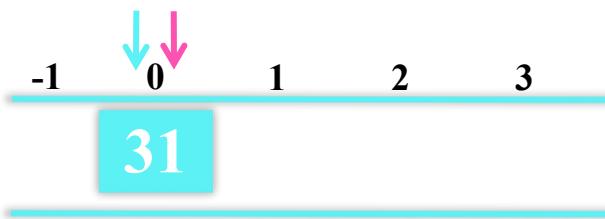
Front Pointer

Rear Pointer

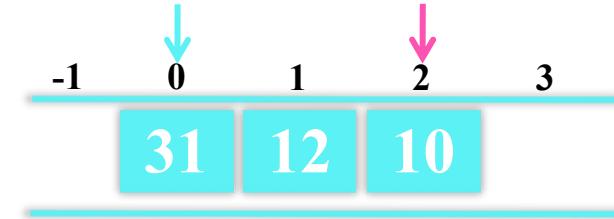
Empty Queue



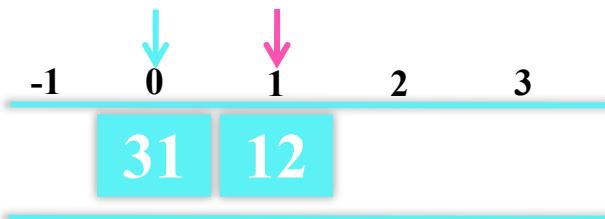
Full Queue



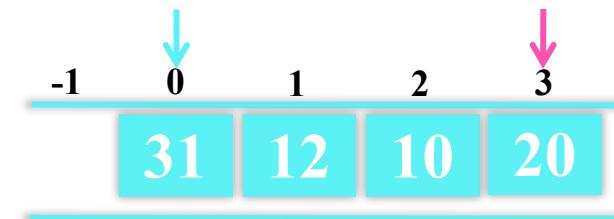
Enqueue(31)



Enqueue(10)



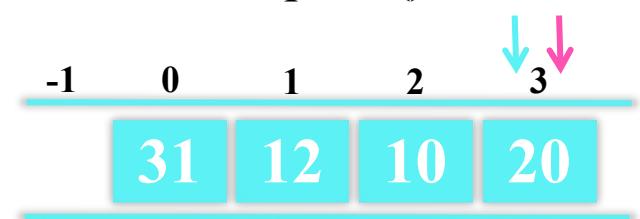
Enqueue(12)



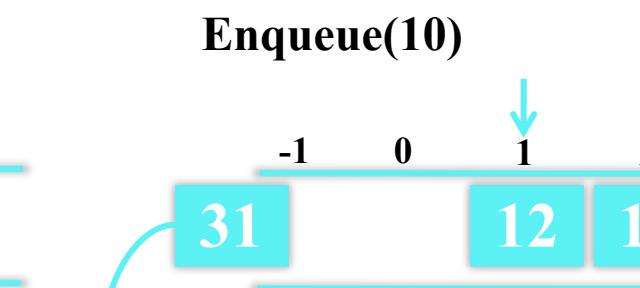
Enqueue(20)



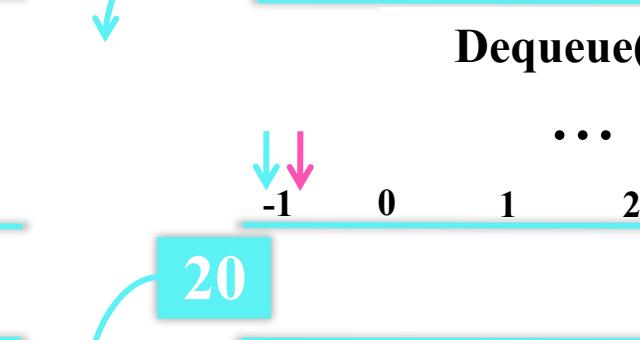
Dequeue()



Underflow



Enqueue(10)



Dequeue()





- **Review: Python Data Structures**
 - Primitive và Non-primitive Data Structures
- **Exercise1: Viết function dùng dictionary data đếm số chữ và từ**
 - Dictionary đếm số chữ trong 1 từ
 - Dictionary đếm số từ trong các câu của 1 file
- **Exercise2: Xây dựng stack data type bằng dictionary và list**
 - Stack với các method cơ bản: CreateStack, IsEmptyStack, IsFullStack, PopStack, PushStack, GetTopStack
- **Exercise3: Xây dựng queue data type bằng dictionary và list**
 - Queue với các method cơ bản: CreateQueue, IsEmptyQueue, IsFullQueue, Dequeue, Enqueue, GetFirstValue

Exercise1: Viết function dùng dictionary data đếm số chữ và từ



1. Thực hiện theo các yêu cầu sau .

(a) Viết function trả về một dictionary **đếm số lượng chữ xuất hiện trong một từ**, với key là chữ cái và value là số lần xuất hiện

- Input: một từ
- Output: dictionary đếm số lần các chữ xuất hiện
- Note: Giả sử các từ nhập vào đều có các chữ cái thuộc [a-z] hoặc [A-Z]

```
1 # Examples 1(a)
2 string = 'Happiness'
3 count_chars(string)
4 >> {'H': 1, 'a': 1, 'e': 1, 'i': 1, 'n': 1, 'p': 2, 's': 2}
5
6 string = 'smiles'
7 count_chars(string)
8 >> {'e': 1, 'i': 1, 'l': 1, 'm': 1, 's': 2}
```

Exercise1: Viết function dùng dictionary data đếm số chữ và từ



1. Thực hiện theo các yêu cầu sau .

(b) Viết function **đọc các câu trong một file txt, đếm số lượng các từ xuất hiện** và trả về một dictionary với key là từ và value là số lần từ đó xuất hiện.

- Input: Đường dẫn đến file txt
- Output: dictionary đếm số lần các từ xuất hiện
- Note:
 - Giả sử các từ trong file txt đều có các chữ cái thuộc [a-z] hoặc [A-Z]
 - Không cần các thao tác xử lý string phức tạp nhưng cần xử lý các từ đều là viết thường
 - Các bạn dùng lệnh này để download và tham khảo Code Listing 2
!gdown <https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko>

```
1 # Examples 1(b)
2 !gdown https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko
3 file_path = '/content/P1_data.txt'
4 word_count(file_path)
5 >>{'a': 7,
6     'again': 1,
7     'and': 1,
8     'are': 1,
9     'at': 1,
10    'be': 1,
11    'become': 2,
12    ...}
```

Code Listing 2: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

Exercise1: Viết function dùng dictionary data đếm số chữ và từ



• Hiểu yêu cầu đề bài

Viết 1 Function

Input là 1 từ bất kỳ

Đếm số chữ trong từ

Output là dictionary chứa
số lần các chữ xuất hiện

Điều kiện cho trước:

- Từ nhập vào luôn có các chữ thuộc [a-z] hoặc [A-Z]

(a) Viết function trả về một dictionary đếm số lượng chữ xuất hiện trong một từ, với key là chữ cái và value là số lần xuất hiện

• Input: một từ

• Output: dictionary đếm số lần các chữ xuất hiện

• Note: Giả sử các từ nhập vào đều có các chữ cái thuộc [a-z] hoặc [A-Z]

Exercise1: Viết function dùng dictionary data đếm số chữ và từ



• Hiểu yêu cầu đề bài

Dùng vòng lặp duyệt từng chữ trong từ

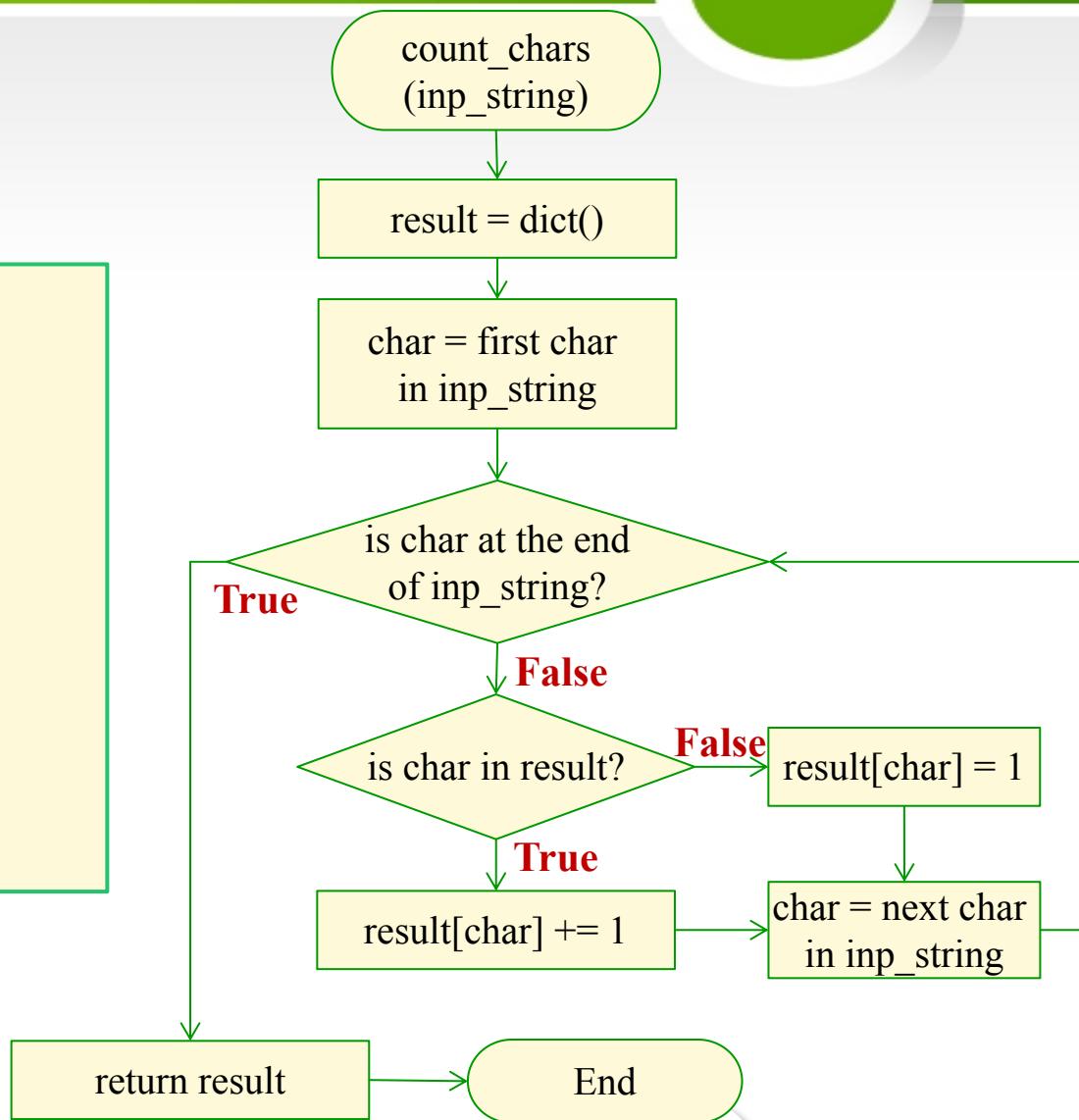
Khởi tạo dictionary rỗng

Nếu chữ xuất hiện lần đầu lấy chữ đó làm key với value = 1

Nếu chữ đã xuất hiện tìm trong dictionary và tăng value lên 1

```
FUNCTION count_chars(inp_string)
    result = dict()
    FOR start at first char TO last char
        IF char in result
            result[char] += 1
        ELSE
            result[char] = 1
        ENDIF
    ENDFOR

    RETURN result
ENDFUNCTION
```



Exercise1: Viết function dùng dictionary data đếm số chữ và từ



• Hiểu yêu cầu đề bài

Viết 1 Function

Input là 1 đường dẫn file txt

Đọc file và lấy các
line trong file

Đếm các từ trong file

Output là dictionary chứa số
lần các từ xuất hiện trong file

(b) Viết function đọc các câu trong một file txt, đếm số lượng các từ xuất hiện và trả về một dictionary với key là từ và value là số lần từ đó xuất hiện

- Input: Đường dẫn đến file txt

- Output: dictionary đếm số lần các từ xuất hiện

- Note:

- Giả sử các từ trong file txt đều có các chữ cái thuộc [a-z] hoặc [A-Z]
- Không cần các thao tác xử lý string phức tạp nhưng cần xử lý các từ đều là viết thường

- Điều kiện cho trước: các từ trong file luôn có
các chữ thuộc [a-z] hoặc [A-Z]

- Yêu cầu: xử lý để các chữ là dạng chữ thường



Exercise1: Viết function dùng dictionary data đếm số chữ và từ



• Hiểu yêu cầu đề bài

Đọc file và lấy các line

Khởi tạo dictionary rỗng

Lặp từng line trong
list line lấy được từ file

Lặp từng từ trong line

Biến đổi thành dạng viết
thường và tách từ trong line

Nếu từ xuất hiện lần đầu lấy
từ đó làm key với value = 1

Nếu từ đã xuất hiện tìm trong
dictionary và tăng value lên 1

```
FUNCTION count_words(file_path)
    OPENFILE file_path for read mode
    OBTAIN all lines FROM read file as a list
    result = dict()
    FOR start at begin of list lines TO end of
        list lines
        FOR start at first word TO the word end
            of the line
            word = word.lower().split(" ")
            IF word in result
                result[word] += 1
            ELSE
                result[word] = 1
            ENDIF
        ENDFOR
    ENDFOR

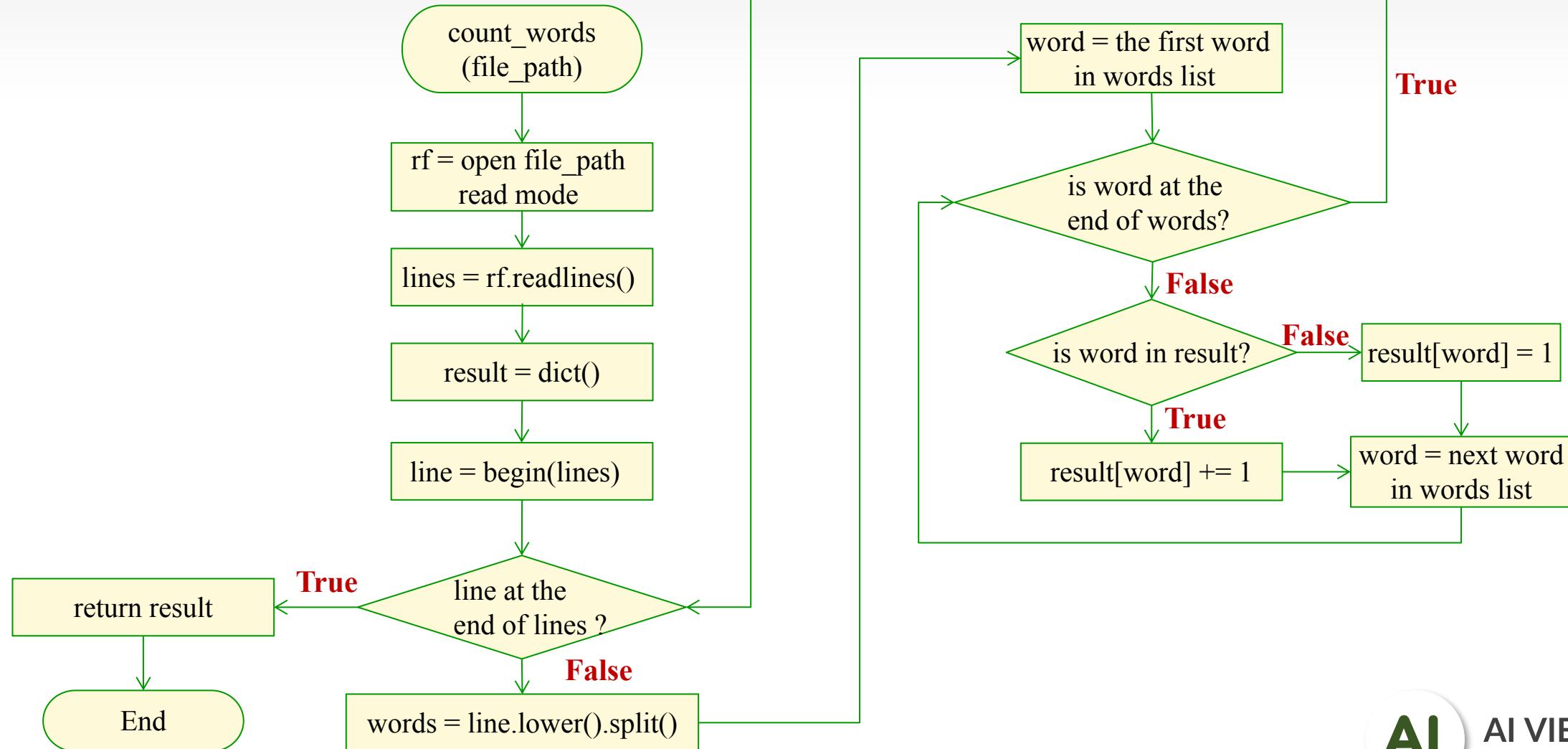
    RETURN result
ENDFUNCTION
```



Exercise1: Viết function dùng dictionary data đếm số chữ và từ



- Hiểu yêu cầu đề bài





- **Review: Python Data Structures**
 - Primitive và Non-primitive Data Structures
- **Exercise1: Viết function dùng dictionary data đếm số chữ và từ**
 - Dictionary đếm số chữ trong 1 từ
 - Dictionary đếm số từ trong các câu của 1 file
- **Exercise2: Xây dựng stack data type bằng dictionary và list**
 - Stack với các method cơ bản: CreateStack, IsEmptyStack, IsFullStack, PopStack, PushStack, GetTopStack
- **Exercise3: Xây dựng queue data type bằng dictionary và list**
 - Queue với các method cơ bản: CreateQueue, IsEmptyQueue, IsFullQueue, Dequeue, Enqueue, GetFirstValue

Exercise2: Xây dựng stack data type bằng dictionary và list



Thực hiện xây dựng Stack bằng dictionary và list data type. Viết một function nhận 4 arguments: method_name, stack, capacity, value để mô phỏng lại cách hoạt động của stack

- **Input:** method_name, stack, capacity, value
- **Outcome:** thực hiện theo method_name và ghi kết quả vào stack (input ở trên)
- Các loại method cần thực hiện trong method_name argument:
 - "CreateStack" dùng để khởi tạo stack
 - "IsEmptyStack" kiểm tra stack có đang rỗng
 - "IsFullStack" kiểm tra stack đã full chưa
 - "PopStack" loại bỏ top element và đưa giá trị vào "RetVal" trong stack dictionary
 - "PushStack" add thêm value (input argument) vào trong stack
 - "GetTopStack" lấy giá trị của top element trong stack đưa vào "TopValue" (không làm thay đổi stack)

```
1 # Examples stack dictionary
2 stack1 = {
3     'Stack' : [1, 2],
4     'Capacity' : 5,
5     'IsEmptyStack' : False,
6     'IsFullStack' : False,
7     'RetVal' : 3,
8     'TopValue' : 4
9 }
10
```

Exercise2: Xây dựng stack data type bằng dictionary và list



```
1 # Examples
2 stack1 = {
3     'Stack' : [],
4     'Capacity' : None,
5     'TopIdx' : -1,
6     'IsEmptyStack' : None,
7     'IsFullStack' : None,
8     'RetValue' : None,
9     'TopValue' : None
10}
11
12
13 simStack(stack=stack1, method_name="CreateStack", capacity=5)
14
15 # stack1.push(1)
16 simStack(stack=stack1, method_name="PushStack", value=1)
17
18 # stack1.push(2)
19 simStack(stack=stack1, method_name="PushStack", value=2)
20
21 # print(stack1.IsFullStack())
22 simStack(stack=stack1, method_name="IsFullStack")
23 print(stack1["IsFullStack"])
24 >> False
25
```

```
26 # print(stack1.top())
27 simStack(stack=stack1, method_name="GetTopStack")
28 print(stack1["TopValue"])
29 >> 2
30
31 # print(stack1.pop())
32 simStack(stack=stack1, method_name="PopStack")
33 print(stack1["RetValue"])
34 >> 2
35
36 # print(stack1.top())
37 simStack(stack=stack1, method_name="GetTopStack")
38 print(stack1["TopValue"])
39 >> 1
40
41 # print(stack1.pop())
42 simStack(stack=stack1, method_name="PopStack")
43 print(stack1["RetValue"])
44 >> 1
45
46 # print(stack1.IsEmptyStack())
47 simStack(stack=stack1, method_name="IsEmptyStack")
48 print(stack1["IsEmptyStack"])
49 >> True
```

Exercise1: Viết function dùng dictionary data đếm số chữ và từ



• Hiểu yêu cầu đề bài

Viết 1 Function

Xây dựng Stack bằng
dictionary và list

Input 4 arguments

Mô phỏng cách
hoạt động của stack

In-place operations

Yêu cầu: stack phải có các method sau

- CreateStack
- IsEmptyStack
- IsFullStack
- PopStack
- PushStack
- GetTopStack

Thực hiện xây dựng Stack bằng dictionary và list data type. Viết một function nhận 4 arguments: method_name, stack, capacity, value để mô phỏng lại cách hoạt động của stack

- **Input:** method_name, stack, capacity, value
- **Outcome:** thực hiện theo method_name và ghi kết quả vào stack (input ở trên)
- Các loại method cần thực hiện trong method_name argument:
 - "CreateStack" dùng để khởi tạo stack
 - "IsEmptyStack" kiểm tra stack có đang rỗng
 - "IsFullStack" kiểm tra stack đã full chưa
 - "PopStack" loại bỏ top element và đưa giá trị vào "RetVal" trong stack dictionary
 - "PushStack" add thêm value (input argument) vào trong stack
 - "GetTopStack" lấy giá trị của top element trong stack đưa vào "TopValue" (không làm thay đổi stack)



Exercise1: Viết function dùng dictionary data đếm số chữ và từ

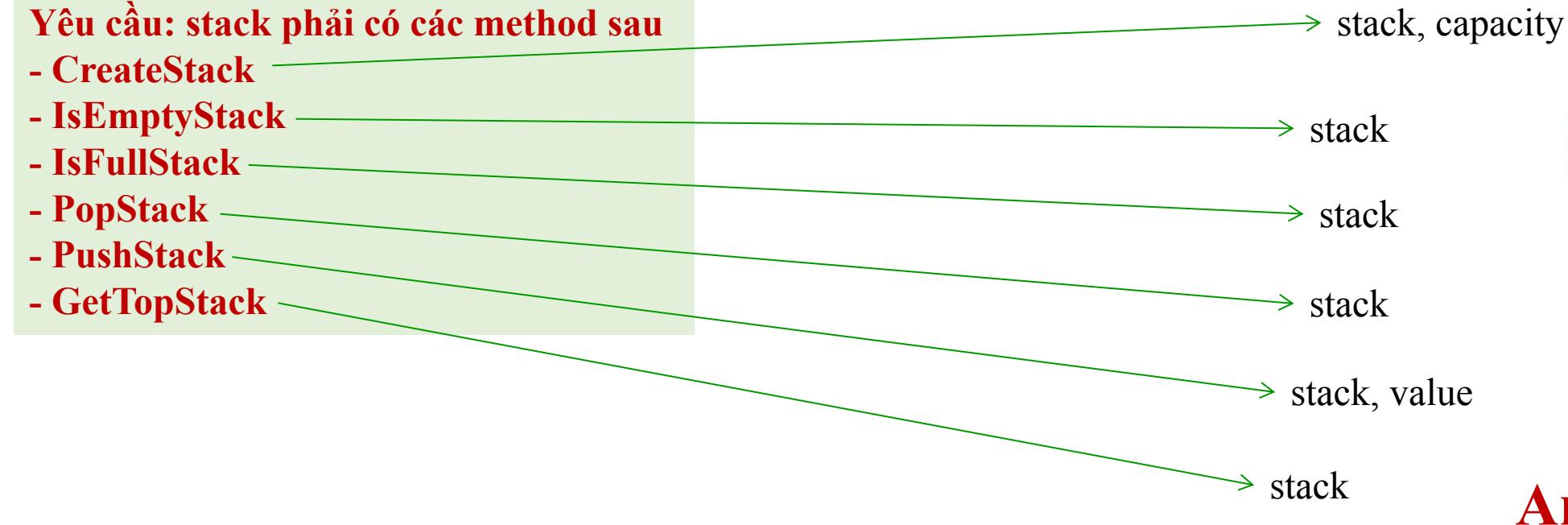


- **Hiểu yêu cầu đề bài**

Input: method_name, stack, capacity, value

Yêu cầu: stack phải có các method sau

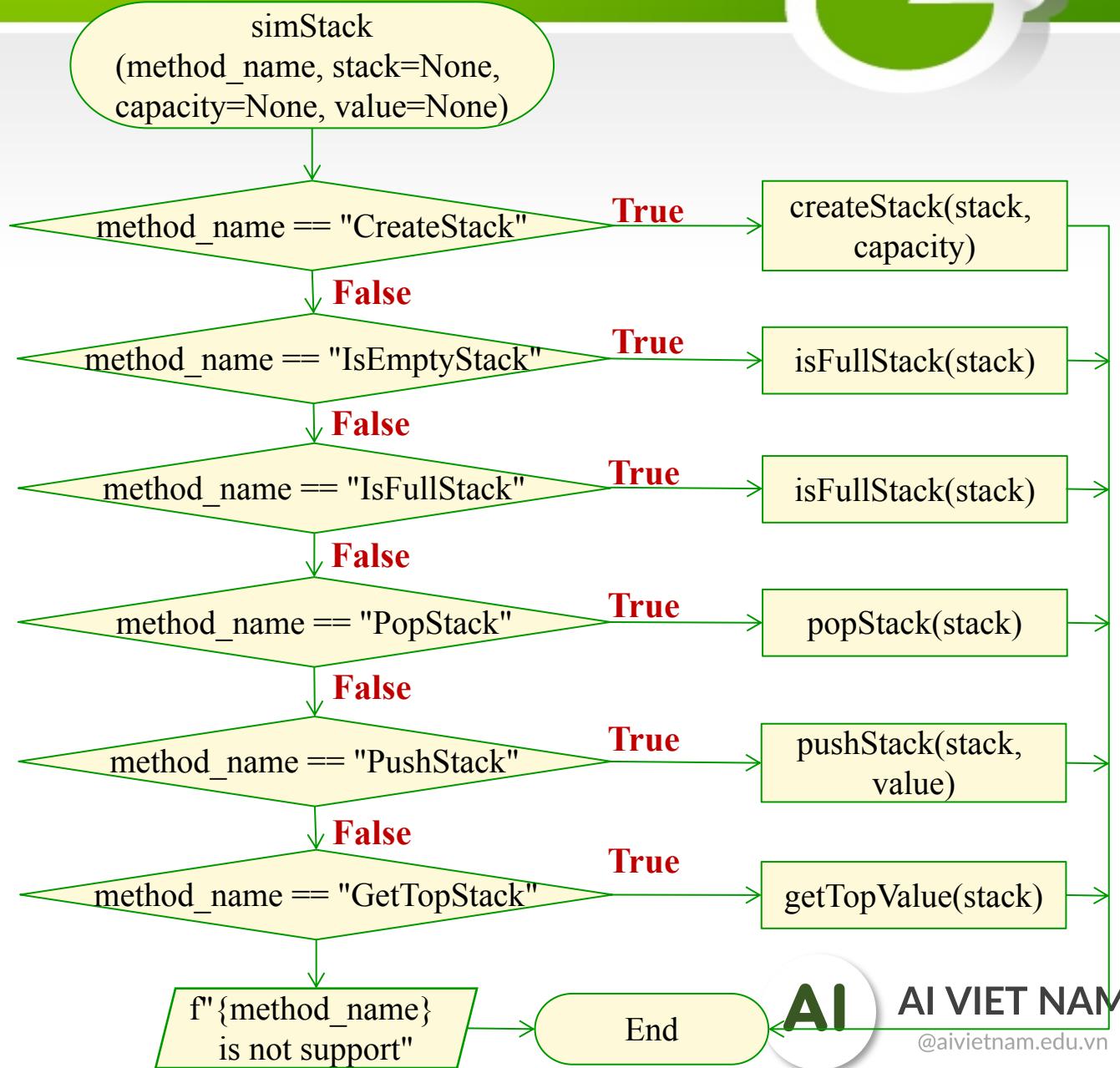
- CreateStack
- IsEmptyStack
- IsFullStack
- PopStack
- PushStack
- GetTopStack



Arguments ?

Exercise1: Viết function dùng dictionary data đếm số chữ và từ

```
FUNCTION simStack(method_name, stack=None,  
                  capacity=None, value=None)  
  
    IF method_name == "CreateStack"  
        createStack(stack, capacity)  
  
    ELIF method_name == "IsEmptyStack"  
        isEmptyStack(stack)  
  
    ELIF method_name == "IsFullStack"  
        isFullStack(stack)  
  
    ELIF method_name == "PopStack"  
        popStack(stack)  
  
    ELIF method_name == "PushStack"  
        pushStack(stack, value)  
  
    ELIF method_name == "GetTopStack"  
        getTopValue(stack)  
  
    ELSE:  
        PRINT f'{method_name} is not support'  
    ENDIF  
ENDFUNCTION
```



Exercise1: Viết function dùng dictionary data đếm số chữ và từ



```
1 # Examples stack dictionary
2 stack1 = {
3     'Stack' : [1, 2],
4     'Capacity' : 5,
5     'IsEmptyStack' : False,
6     'IsFullStack' : False,
7     'RetVal' : 3,
8     'TopValue' : 4
9 }
10 }
```



Initial values ?

```
1 # Examples
2 stack1 = {
3     'Stack' : [],
4     'Capacity' : None,
5     'TopIdx' : -1,
6     'IsEmptyStack' : None,
7     'IsFullStack' : None,
8     'RetVal' : None,
9     'TopValue' : None
10 }
```

FUNCTION createStack(stack, capacity)

IF type(stack) is not type(dict())
PRINT 'CreateStack method expected stack argument'
RETURN

ENDIF

IF capacity is None
PRINT 'CreateStack method expected capacity argument'
RETURN

ENDIF

stack['Stack'] = list()
stack['Capacity'] = capacity
stack['TopIdx'] = -1
stack['IsEmptyStack'] = None
stack['IsFullStack'] = None
stack['RetVal'] = None
stack['TopValue'] = None

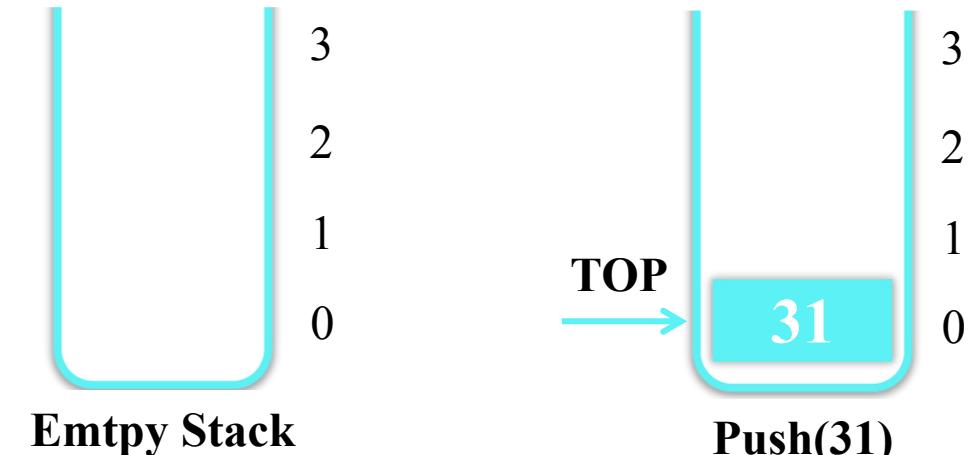
ENDFUNCTION

Exercise1: Viết function dùng dictionary data đếm số chữ và từ

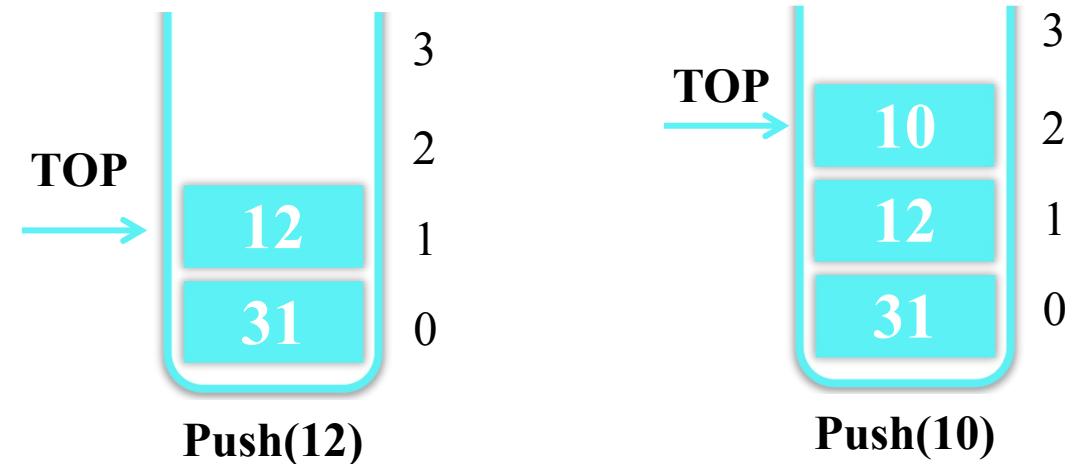


```
FUNCTION isEmptyStack(stack)
    IF type(stack) is not type(dict())
        PRINT 'IsEmptyStack method expected stack argument'
        RETURN
    ENDIF
    IF stack['TopIdx'] == -1
        stack['IsEmptyStack'] = True
    ELSE
        stack['IsEmptyStack'] = False
    ENDFUNCTION
```

Capacity = 4



TopIdx ?



Exercise1: Viết function dùng dictionary data đếm số chữ và từ

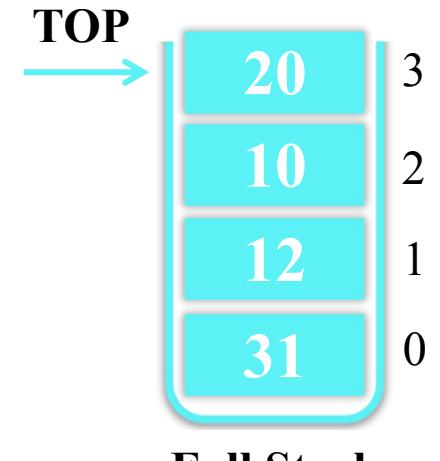
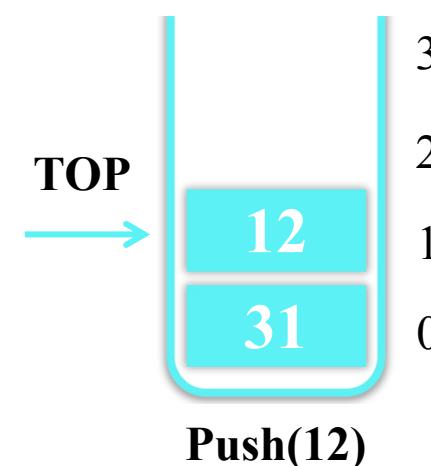
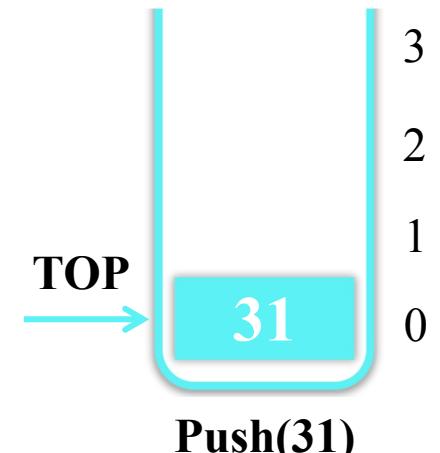


```
FUNCTION IsFullStack(stack)
    IF type(stack) is not type(dict())
        PRINT 'IsFullStack method expected stack argument'
        RETURN
    ENDIF
    IF stack['TopIdx'] == stack['Capacity']-1
        stack['IsFullStack'] = True
    ELSE:
        stack['IsFullStack'] = False
    ENDFUNCTION
```



TopIdx ?

Capacity = 4



Exercise1: Viết function dùng dictionary data đếm số chữ và từ

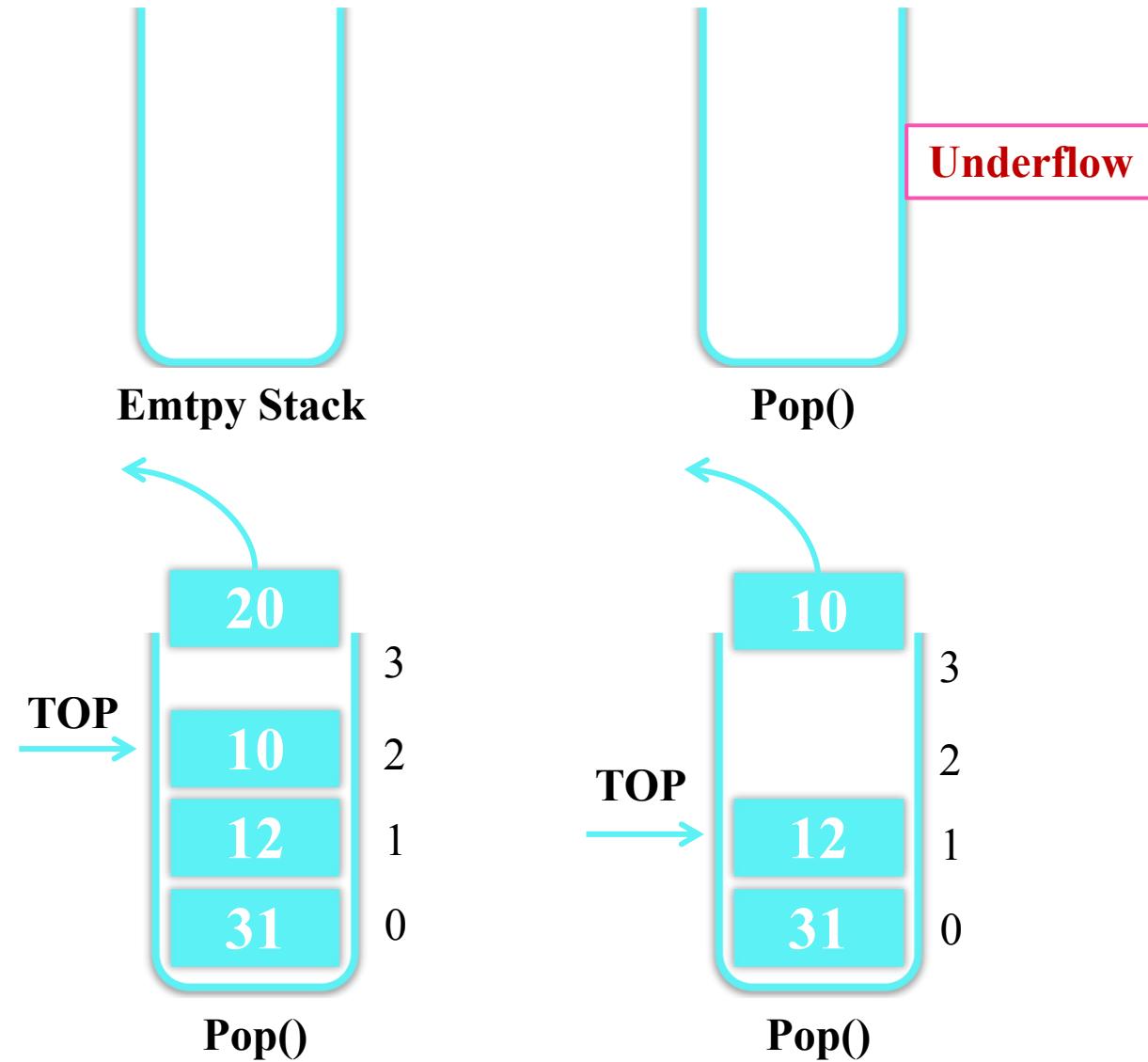
```
FUNCTION popStack(stack)
    IF type(stack) is not type(dict())
        PRINT 'PopStack method expected stack argument'
        RETURN
    ENDIF

    isEmptyStack(stack)

    IF stack['IsEmptyStack']
        PRINT "Stack is empty"
        RETURN
    ELSE
        stack['RetValue'] = stack['Stack'].pop()
        stack['TopIdx'] -= 1
    ENDIFUNCTION
```



Exception ?



Exercise1: Viết function dùng dictionary data đếm số chữ và từ

```
FUNCTION pushStack(stack, value)
```

```
    IF type(stack) is not type(dict())
```

```
        PRINT 'PushStack method expected stack argument'
```

```
        RETURN
```

```
ENDIF
```

```
IF value is None
```

```
    PRINT 'PushStack method expected value argument"
```

```
    RETURN
```

```
ENDIF
```

```
isFullStack(stack)
```

```
IF stack['IsFullStack']
```

```
    PRINT "Stack is full"
```

```
    RETURN
```

```
ELSE
```

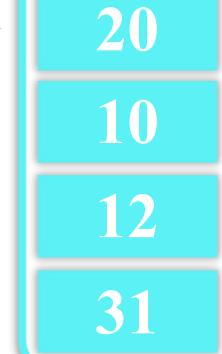
```
    stack['Stack'].append(value)
```

```
    stack['TopIdx'] += 1
```

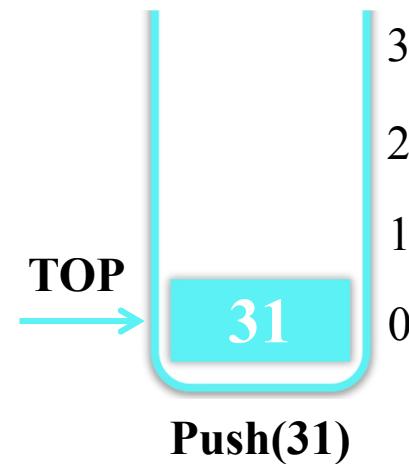
```
ENDFUNCTION
```

Capacity = 4

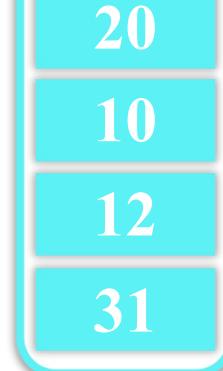
TOP



Exception ?

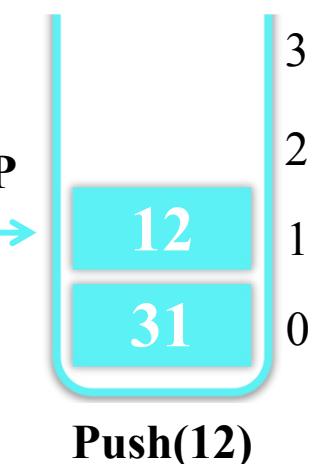


TOP



Push(20)

TOP



Push(12)



AI VIET NAM

@avietnam.edu.vn

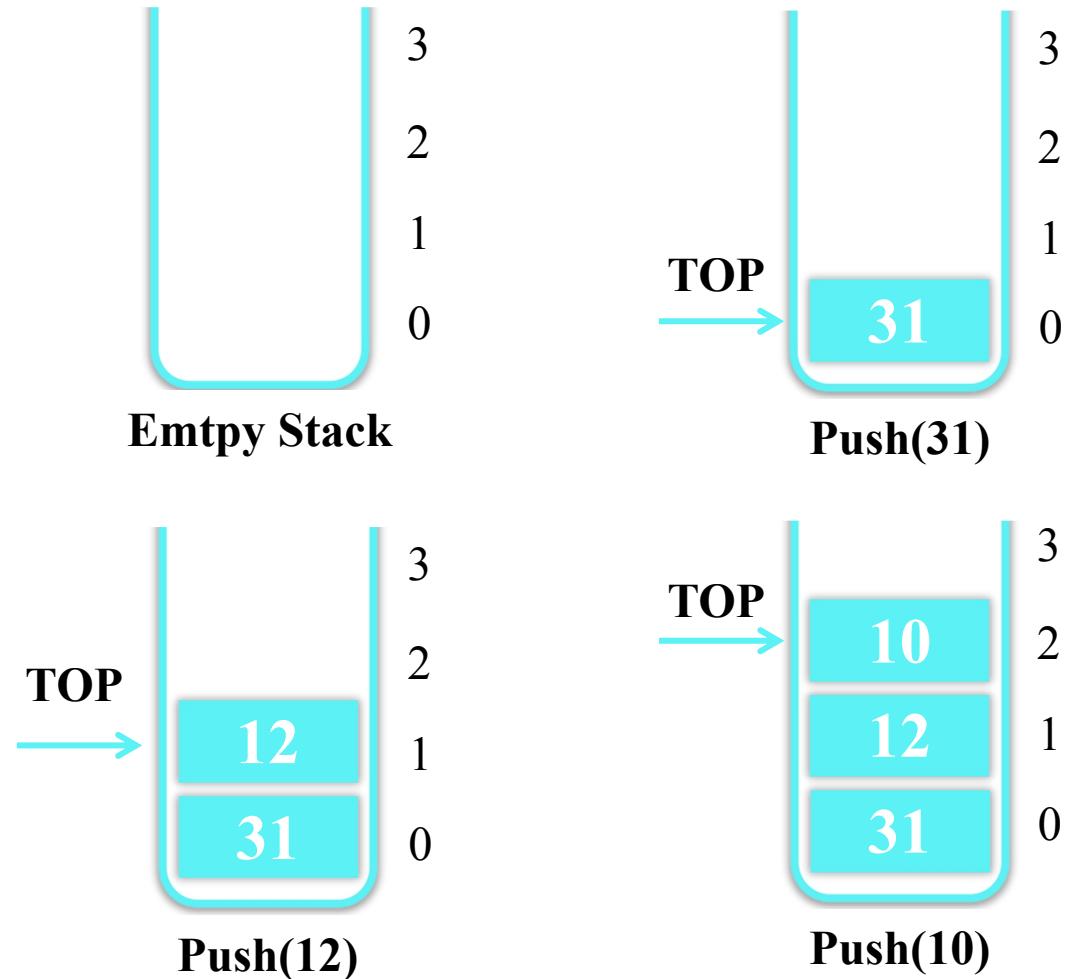
Exercise1: Viết function dùng dictionary data đếm số chữ và từ



```
FUNCTION getTopValue(stack)
    IF type(stack) is not type(dict())
        PRINT 'GetTopValue method expected stack argument'
        RETURN
    ENDIF

isEmptyStack(stack)

    IF stack['IsEmptyStack']:
        PRINT "Stack is empty"
        RETURN
    ELSE
        stack['TopValue'] = stack['Stack'][stack['TopIdx']]
    ENDFUNCTION
```





- **Review: Python Data Structures**
 - Primitive và Non-primitive Data Structures
- **Exercise1: Viết function dùng dictionary data đếm số chữ và từ**
 - Dictionary đếm số chữ trong 1 từ
 - Dictionary đếm số từ trong các câu của 1 file
- **Exercise2: Xây dựng stack data type bằng dictionary và list**
 - Stack với các method cơ bản: CreateStack, IsEmptyStack, IsFullStack, PopStack, PushStack, GetTopStack
- **Exercise3: Xây dựng queue data type bằng dictionary và list**
 - Queue với các method cơ bản: CreateQueue, IsEmptyQueue, IsFullQueue, Dequeue, Enqueue, GetFirstValue

Exercise3: Xây dựng queue data type bằng dictionary và list



Thực hiện xây dựng Queue bằng dictionary và list data type. Viết một function nhận 4 arguments: method_name, queue, capacity, và value để mô phỏng lại cách hoạt động của queue

- **Input:** method_name, queue, capacity, value
- **Outcome:** thực hiện theo method_name và ghi kết quả vào queue (input ở trên)
- Các loại method cần thực hiện trong method_name argument:
 - "CreateQueue" dùng để khởi tạo queue
 - "IsEmptyQueue" kiểm tra queue có đang rỗng
 - "IsFullQueue" kiểm tra queue đã full chưa
 - "Dequeue" loại bỏ first element và đưa giá trị vào "RetVal" trong queue dictionary
 - "Enqueue" add thêm value (input argument) vào trong queue
 - "GetFirstValue" lấy giá trị của first element trong queue đưa vào "FirstValue" (không làm thay đổi queue)

```
1 # Examples queue dictionary
2 queue1 = {
3     'Queue' : [],
4     'Capacity' : None,
5     'IsEmptyQueue' : None,
6     'IsFullQueue' : None,
7     'RetVal' : None,
8     'FirstValue' : None
9 }
10
```



Exercise3: Xây dựng queue data type bằng dictionary và list



```
1 # Example
2 queue1 = {
3     'Queue' : [],
4     'Capacity' : None,
5     'IsEmptyQueue' : None,
6     'IsFullQueue' : None,
7     'RetValue' : None,
8     'FirstValue' : None
9 }
10
11 simQueue(queue=queue1, method_name="CreateQueue", capacity=5)
12
13 # queue1.enqueue(1)
14 simQueue(queue=queue1, method_name="Enqueue", value=1)
15
16 # queue1.enqueue(2)
17 simQueue(queue=queue1, method_name="Enqueue", value=2)
18
19 # print(queue1.IsFullQueue())
20 simQueue(queue=queue1, method_name="IsFullQueue")
21 print(queue1["IsFullQueue"])
22 >> False
23
24 # print(queue1.front())
25 simQueue(queue=queue1, method_name="GetFirstValue")
26 print(queue1["FirstValue"])
27 >> 1
```

```
28
29 # print(queue1.dequeue())
30 simQueue(queue=queue1, method_name="Dequeue")
31 print(queue1["RetValue"])
32 >> 1
33
34 # print(queue1.front())
35 simQueue(queue=queue1, method_name="GetFirstValue")
36 print(queue1["FirstValue"])
37 >> 2
38
39 # print(queue1.dequeue())
40 simQueue(queue=queue1, method_name="Dequeue")
41 print(queue1["RetValue"])
42 >> 2
43
44 # print(queue1.IsEmptyQueue())
45 simQueue(queue=queue1, method_name="IsEmptyQueue")
46 print(queue1["IsEmptyQueue"])
47 >> True
```

Exercise3: Xây dựng queue data type bằng dictionary và list



• Hiểu yêu cầu đề bài

Viết 1 Function

Xây dựng Stack bằng
dictionary và list

Input 4 arguments

Mô phỏng cách
hoạt động của stack

In-place operations

Yêu cầu: stack phải có các method sau

- CreateQueue
- IsEmptyQueue
- IsFullQueue
- Dequeue
- Enqueue
- GetFirstValue

Thực hiện **xây dựng Queue bằng dictionary và list** data type. Viết một function nhận 4 arguments: **method_name**, **queue**, **capacity**, và **value** để mô phỏng lại cách hoạt động của queue

- **Input:** method_name, queue, capacity, value
- **Outcome:** thực hiện theo method_name và ghi kết quả vào queue(input ở trên)
- Các loại method cần thực hiện trong method_name argument:
 - "CreateQueue" dùng để khởi tạo queue
 - "IsEmptyQueue" kiểm tra queue có đang rỗng
 - "IsFullQueue" kiểm tra queue đã full chưa
 - "Dequeue" loại bỏ first element và đưa giá trị vào "RetVal" trong queue dictionary
 - "Enqueue" add thêm value (input argument) vào trong queue
 - "GetFirstValue" lấy giá trị của first element trong queue đưa vào "FirstValue" (không làm thay đổi queue)



Exercise3: Xây dựng queue data type bằng dictionary và list

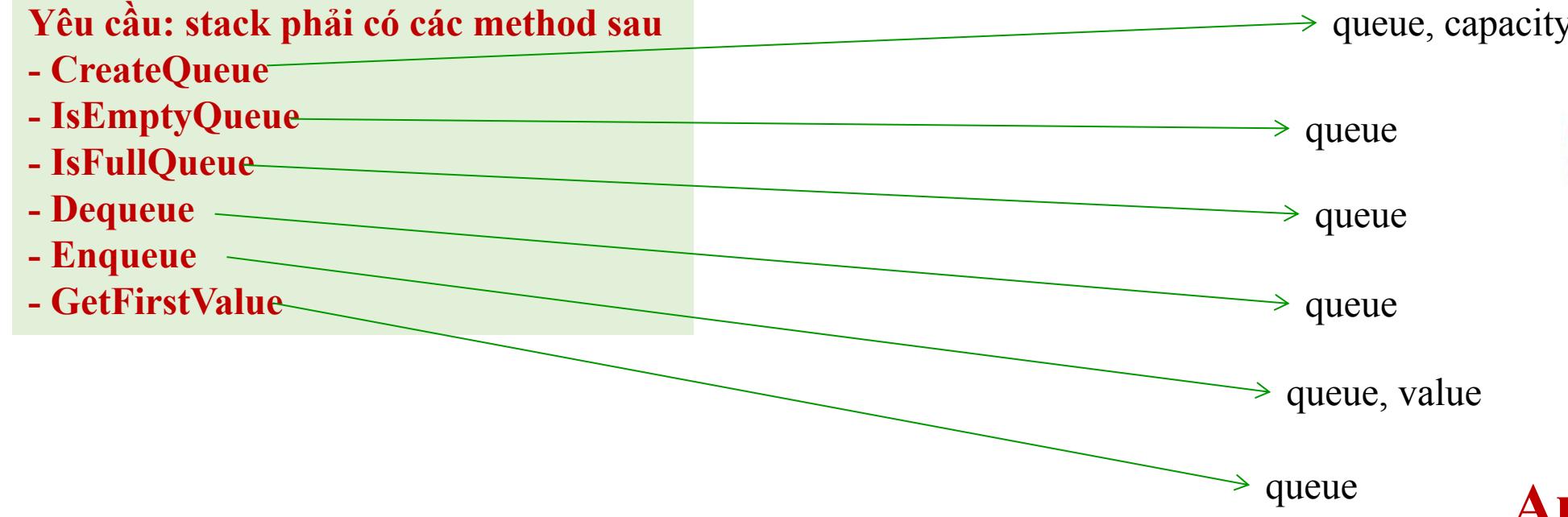


- **Hiểu yêu cầu đề bài**

Input: method_name, queue, capacity, value

Yêu cầu: stack phải có các method sau

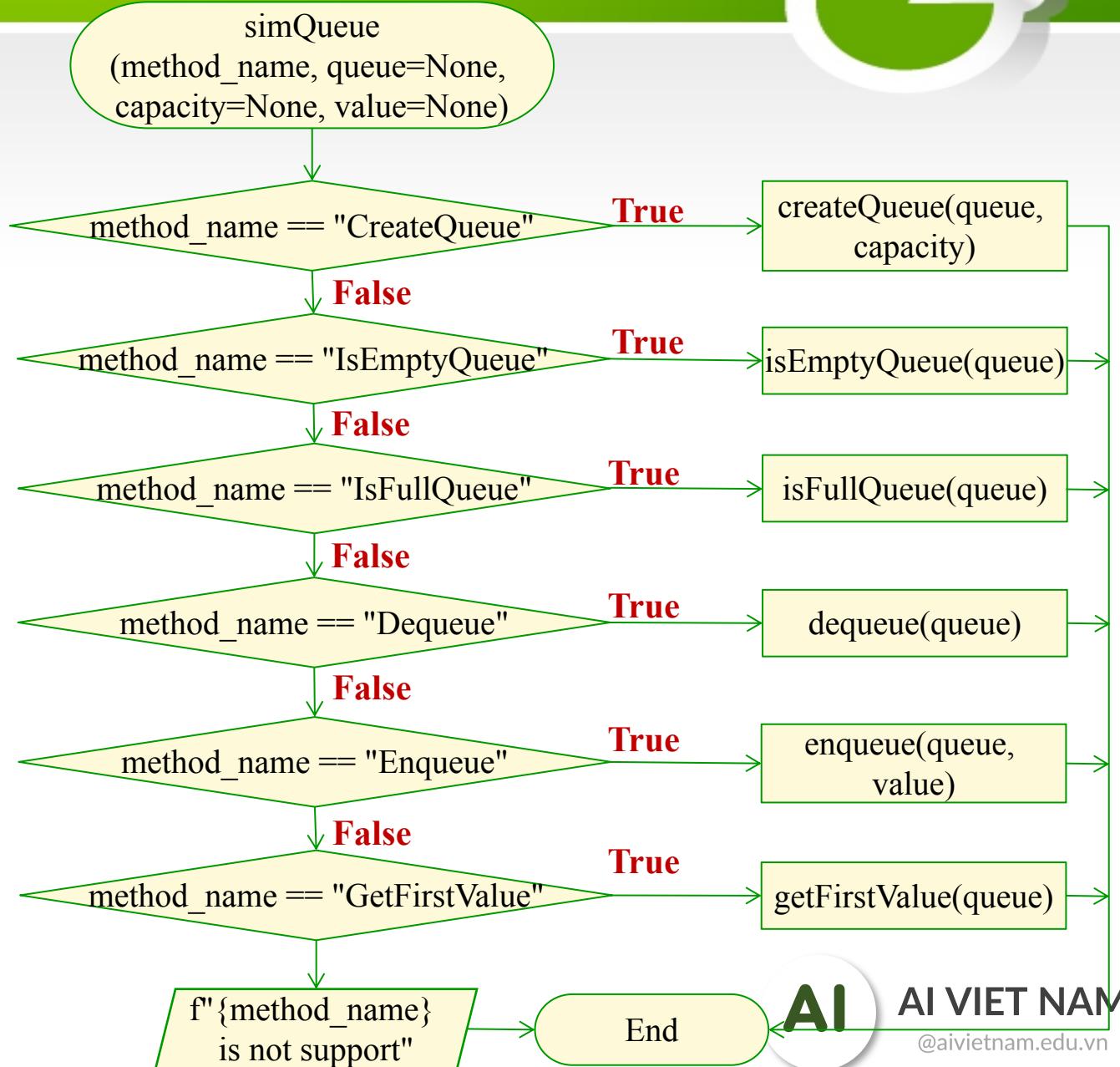
- CreateQueue
- IsEmptyQueue
- IsFullQueue
- Dequeue
- Enqueue
- GetFirstValue



Arguments ?

Exercise3: Xây dựng queue data type bằng dictionary và list

```
FUNCTION simQueue(method_name, queue=None,  
                  capacity=None, value=None)  
  
    IF method_name == "CreateQueue":  
        createQueue(queue, capacity)  
  
    ELIF method_name == "IsEmptyQueue"  
        isEmptyQueue(queue)  
  
    ELIF method_name == "IsFullQueue"  
        isFullQueue(queue)  
  
    ELIF method_name == "Dequeue"  
        dequeue(queue)  
  
    ELIF method_name == "Enqueue"  
        enqueue(queue, value)  
  
    ELIF method_name == "GetFirstValue"  
        getFirstValue(queue)  
  
    ELSE:  
        PRINT f'{method_name} is not support'  
    ENDIF  
ENDFUNCTION
```



Exercise3: Xây dựng queue data type bằng dictionary và list



```
1 # Example
2 queue1 = {
3     'Queue' : [],
4     'Capacity' : None,
5     'IsEmptyQueue' : None,
6     'IsFullQueue' : None,
7     'RetValue' : None,
8     'FirstValue' : None
9 }
```

```
1 # Examples queue dictionary
2 queue1 = {
3     'Queue' : [1, 2],
4     'Capacity' : 5,
5     'IsEmptyQueue' : False,
6     'IsFullQueue' : False,
7     'RetValue' : 3,
8     'FirstValue' : 4
9 }
10
```



Initial values ?

```
FUNCTION createQueue(queue, capacity)
    IF type(queue) is not type(dict())
        PRINT 'CreateQueue method expected queue argument'
        RETURN
    ENDIF

    IF capacity is None
        PRINT 'CreateQueue method expected capacity argument'
        RETURN
    ENDIF

    queue['Queue'] = list()
    queue['Capacity'] = capacity
    queue['IsEmptyQueue'] = None
    queue['IsFullQueue'] = None
    queue['RetValue'] = None
    queue['FirstValue'] = None

ENDFUNCTION
```



Exercise3: Xây dựng queue data type bằng dictionary và list



```
FUNCTION isEmptyQueue(queue)
    IF type(queue) is not type(dict())
        PRINT 'IsEmptyQueue method expected queue argument'
        RETURN
    ENDIF

    queue['IsEmptyQueue'] = len(queue['Queue']) == 0

ENDFUNCTION
```

```
FUNCTION isFullQueue(queue)
    IF type(queue) is not type(dict())
        PRINT 'IsFullQueue method expected queue argument'
        RETURN
    ENDIF

    queue['IsFullQueue'] = len(queue['Queue']) == queue['Capacity']

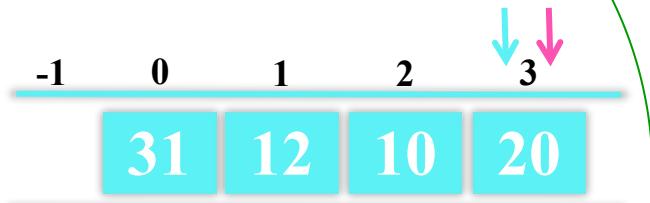
ENDFUNCTION
```

Capacity = 4



Empty Queue

Thực hiện dựa trên list
Check length của list



Full Queue

Exercise3: Xây dựng queue data type bằng dictionary và list



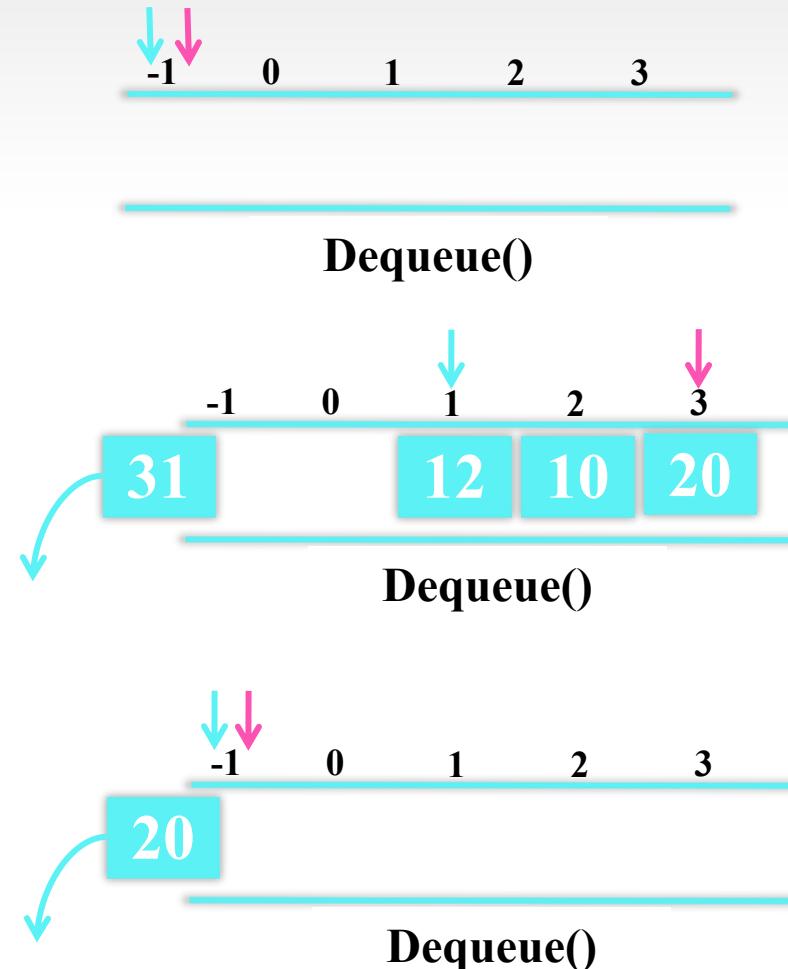
```
FUNCTION dequeue(queue)
    IF type(queue) is not type(dict())
        PRINT 'Dequeue method expected queue argument'
        RETURN
    ENDIF

    isEmptyQueue(queue)

    IF queue['IsEmptyQueue']
        PRINT "Queue is empty"
        RETURN
    ELSE
        queue['RetVal'] = queue['Queue'].pop(0)
    ENDIF
```



Exception ?



Exercise3: Xây dựng queue data type bằng dictionary và list



FUNCTION enqueue(queue, value)

IF type(queue) is not type(dict())

PRINT 'Enqueue method expected queue argument'

RETURN

ENDIF

IF value is None

PRINT 'Enqueue method expected value argument'

RETURN

ENDIF

 isFullQueue(queue)

IF queue['IsFullQueue']

PRINT "Queue is full"

RETURN

ELSE

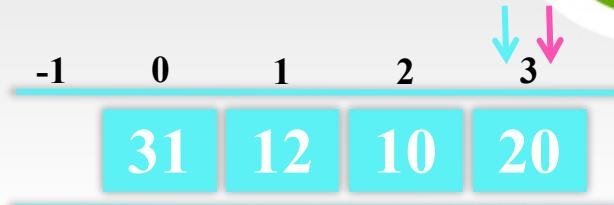
 queue['Queue'].append(value)

ENDFUNCTION

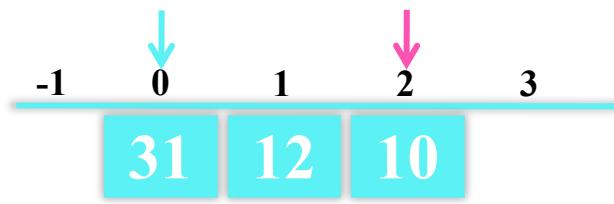
Capacity = 4



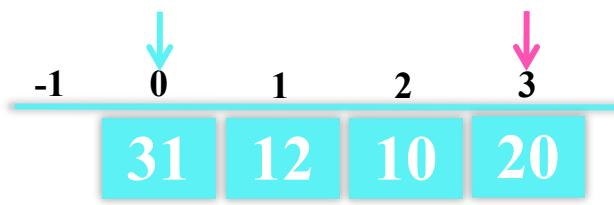
Exception ?



Enqueue(10)



Enqueue(10)



Enqueue(20)

Overflow

Exercise3: Xây dựng queue data type bằng dictionary và list



```
FUNCTION getFirstValue(queue)
    IF type(queue) is not type(dict())
        PRINT 'GetFirstValue method expected queue argument'
        RETURN
    ENDIF

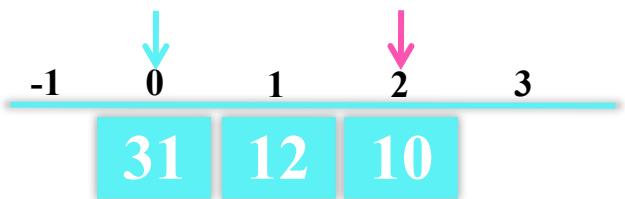
    isEmptyQueue(queue)

    IF queue['IsEmptyQueue']:
        PRINT "Queue is empty"
        RETURN
    ELSE
        queue['FirstValue'] = queue['Queue'][0]
    ENDIFUNCTION
```

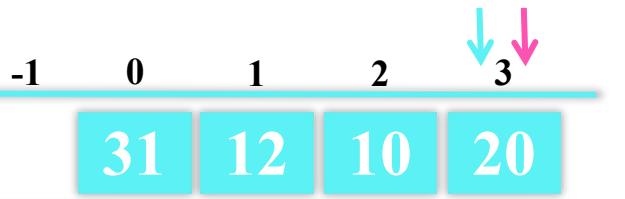
Capacity = 4



Empty Queue



Enqueue(10)



Full Queue



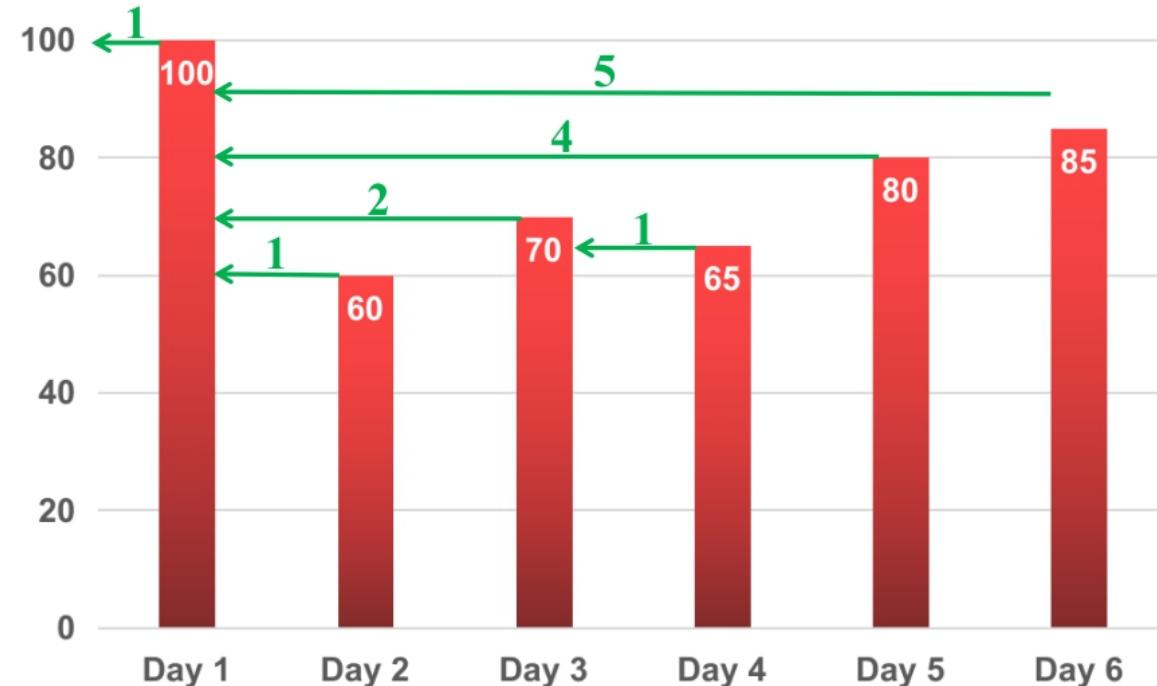
- **Exercise4: Application of Stack - Stock Span Problem**
 - Tính số ngày tối đa liên tiếp từ ngày hiện tại cho đến các ngày trước trong quá khứ có giá cổ phiếu thấp hơn hoặc bằng giá cổ phiếu hiện tại
- **Exercise5: Application of Queue - Generate Binary number from 1 to N**
 - Tạo ra chuỗi binary từ 1 đến N và đưa kết quả vào một dictionary
- **Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA**
 - Tìm và trả về một list các chuỗi có 10 ký tự được lặp lại hơn một lần trong chuỗi DNA cho trước

Exercise4: Application of Stack - Stock Span Problem



Stock Span Problem: Cho trước một list giá của một loại cổ phiếu trong N ngày. Viết function đi tìm stock span cho mỗi ngày trong list trên.

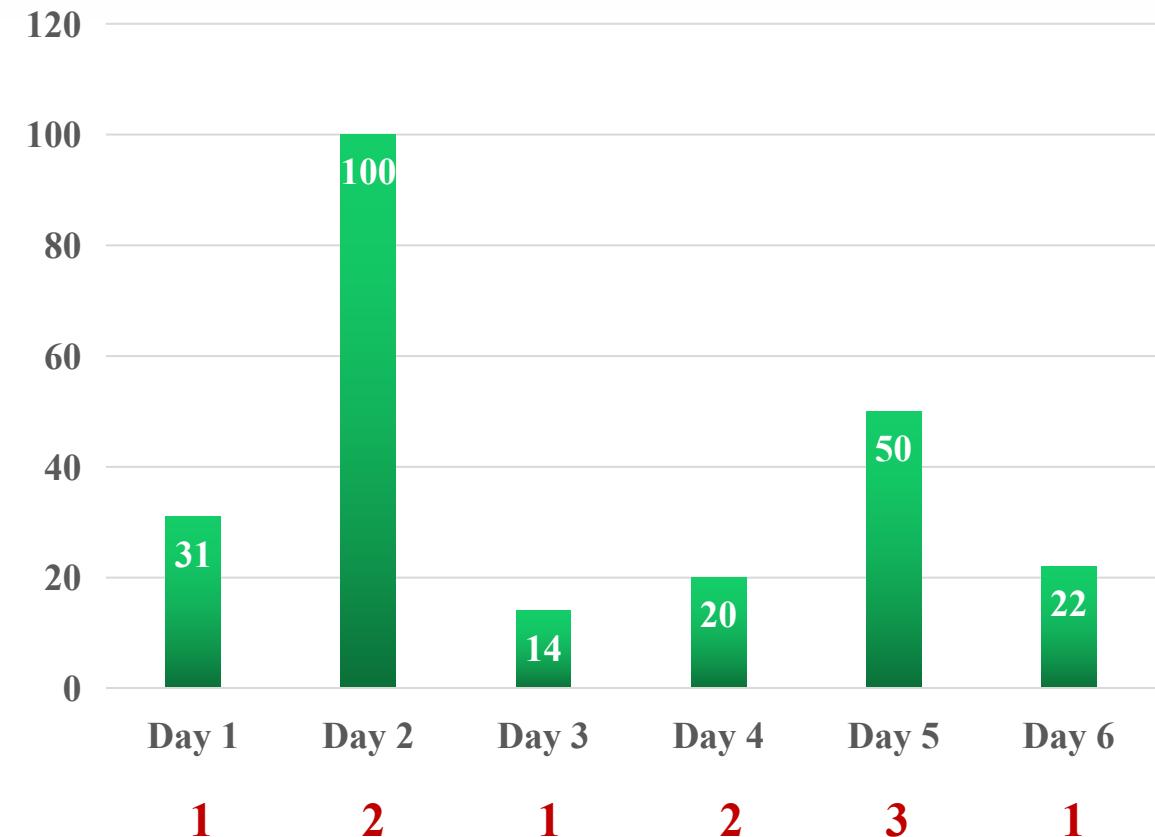
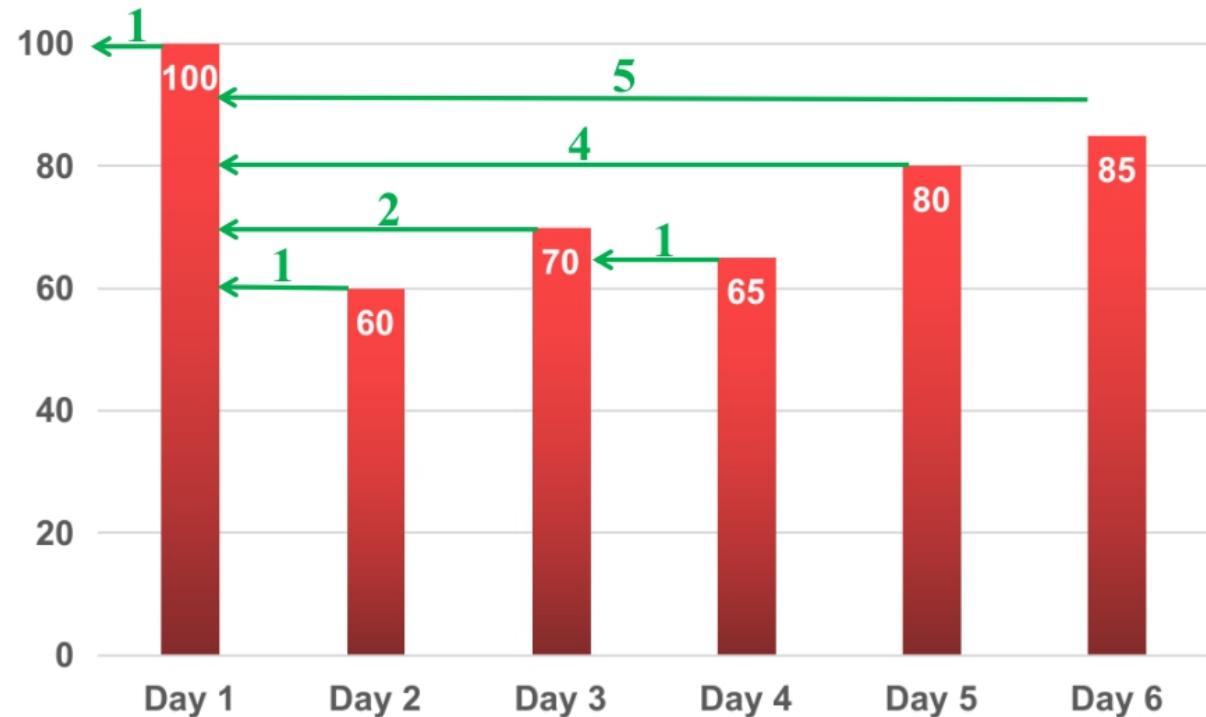
- **Input:** list chứa giá cổ phiếu của N ngày (một ngày là một giá)
- **Output:** trả về một list là stock span của từng ngày
- **Khuyến khích các bạn sử dụng stack, và nếu có thể thì sử dụng stack của các bạn đã thực hiện ở bài 2**
- **Stock Span:** của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại
- **Ví dụ:**
 - Input: [100, 60, 70, 65, 80, 85]
 - Output: [1, 1, 2, 1, 4, 5]



Exercise4: Application of Stack - Stock Span Problem



Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



Exercise4: Application of Stack - Stock Span Problem



• Hiểu yêu cầu đề bài

Viết 1 Function

Input list giá của N ngày

Tìm stock span của mỗi ngày

Output list stock span

Stock Span Problem: Cho trước một list giá của một loại cổ phiếu trong N ngày.
Viết function để tìm stock span cho mỗi ngày trong list trên.

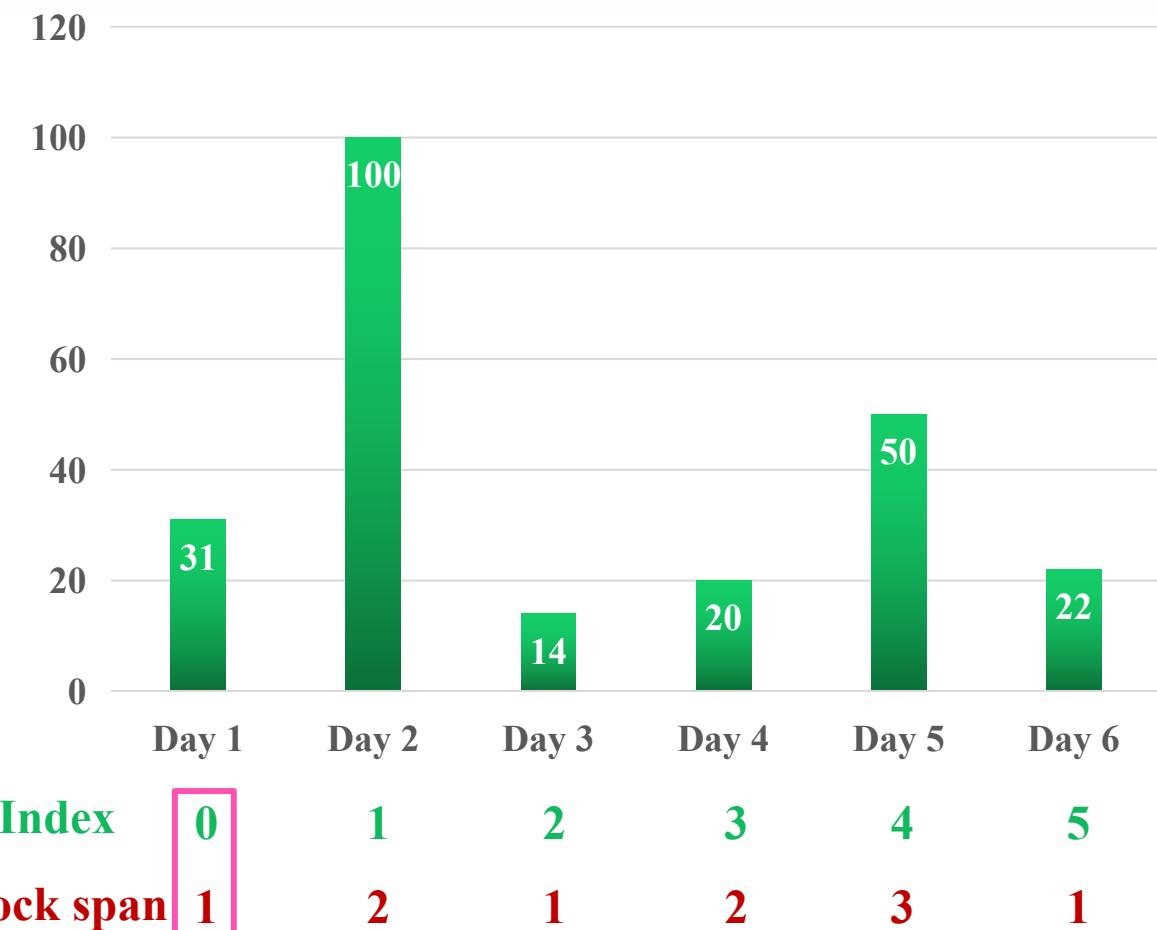
- **Input:** list chứa giá cổ phiếu của N ngày (một ngày là một giá)
- **Output:** trả về một list là stock span của từng ngày
- **Khuyến khích các bạn sử dụng stack, và nếu có thể thì sử dụng stack của các bạn đã thực hiện ở bài 2**

Khuyến khích sử dụng stack

Exercise4: Application of Stack - Stock Span Problem



Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



Gọi t là thời điểm hiện tại:

- $t = \text{Day}4$ quan sát thấy điểm dừng là Day2 (stop) (Day2 lớn hơn Day4, trong khi Day3 bé hơn Day4)
=> stock span = 2 (Day3, Day4)
- $t = \text{Day}5$ quan sát thấy điểm dừng là Day2 (stop) (Day2 lớn Day5, trong khi Day3 và Day4 đều bé hơn Day5)
=> stock span = 3 (Day3, Day4, Day5)
- $t = \text{Day}6$ quan sát thấy điểm dừng là Day5 (stop) (Day5 lớn hơn Day6)
=> stock span = 1 (Day6)

=> stock span = index_t - index_{stop}

VD:

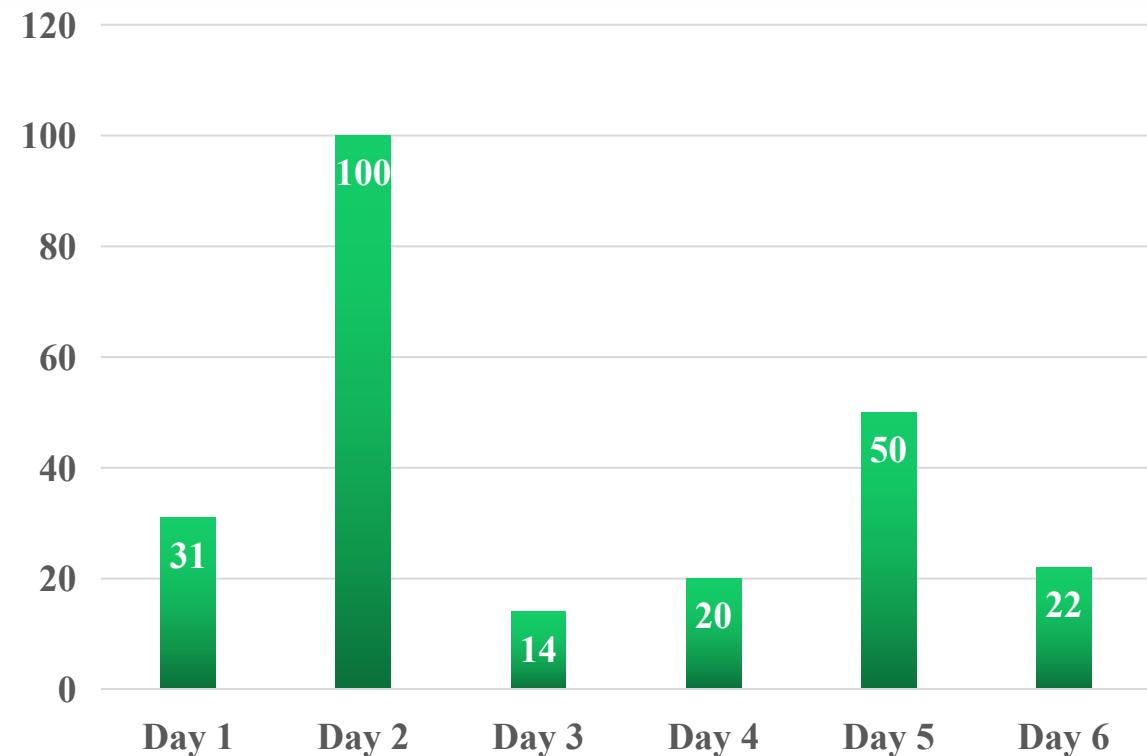
- $t = \text{Day}4 \rightarrow \text{stock span} = 3 - 1 = 2$
- $t = \text{Day}5 \rightarrow \text{stock span} = 4 - 1 = 3$
- $t = \text{Day}6 \rightarrow \text{stock span} = 5 - 4 = 1$



Exercise4: Application of Stack - Stock Span Problem



Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



Gọi t là thời điểm hiện tại:

- $t = \text{Day}2$ quan sát thấy không có điểm dừng (stop) (không có ngày nào có giá trị lớn hơn Day2), trong khi vẫn có Day1 bé hơn Day2

$$\Rightarrow \text{stock span} = 2 \text{ (Day1, Day2)}$$

=> Cách 1 stock span = index_t + 1

VD:

$$- t = \text{Day}4 \rightarrow \text{stock span} = 1 + 1 = 2$$

=> Cách 1 stock span = index_t - (-1)

VD:

$$- t = \text{Day}4 \rightarrow \text{stock span} = 1 - (-1) = 2$$

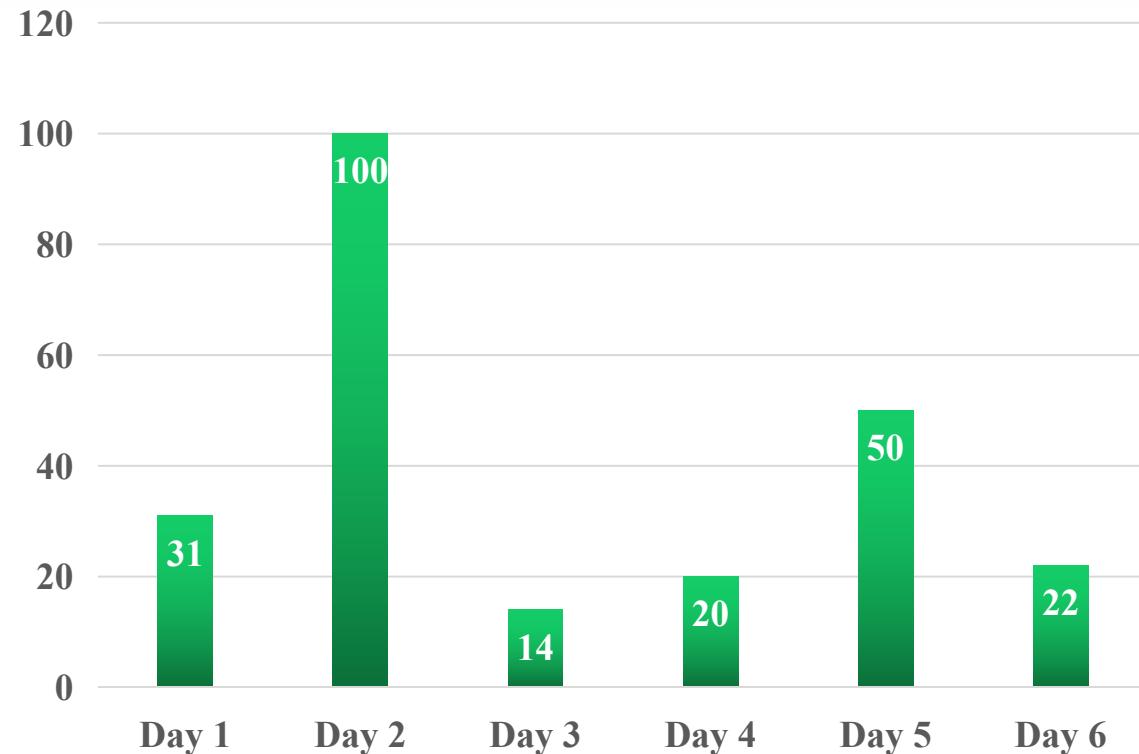
Index	0	1	2	3	4	5
Stock span	1	2	1	2	3	1



Exercise4: Application of Stack - Stock Span Problem



Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



Index	0	1	2	3	4	5
Stock span	1	2	1	2	3	1

Tại thời điểm hiện tại t :

- Stack chứa index các ngày trở về trước có $\text{value}(t') \geq \text{value}(t-1)$ (bao gồm $t-1$)
- Các ngày trong stack có giá trị giảm dần cho đến top

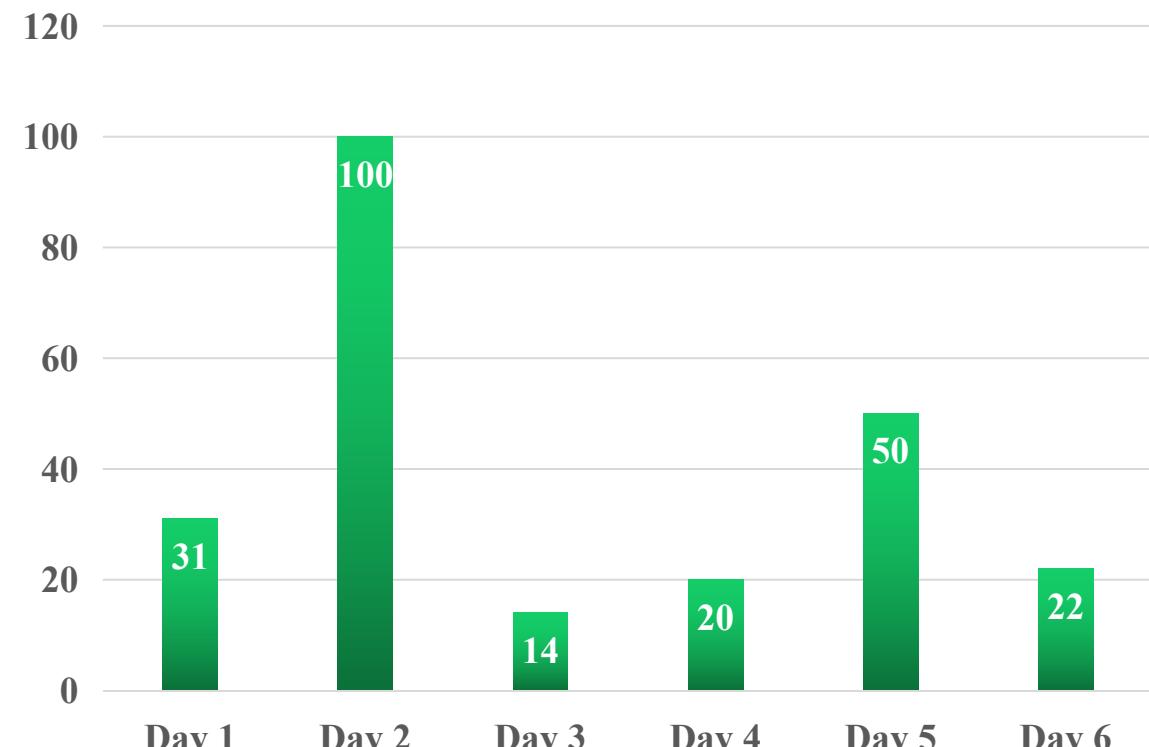
\Rightarrow Chứa stop của $t-1$ và stop của t' và bản thân $t-1$ và giá trị giảm dần



Exercise4: Application of Stack - Stock Span Problem



Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



Index	Stock span
0	1
1	2
2	1
3	2
4	3
5	1

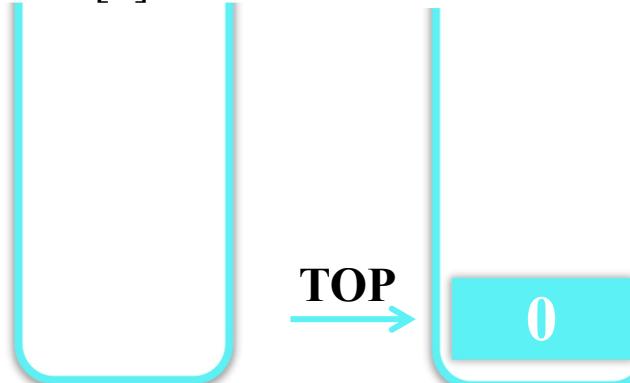
$$\text{stock span} = [1]$$



t = Day1

$$\text{stock span} = [1, 2]$$

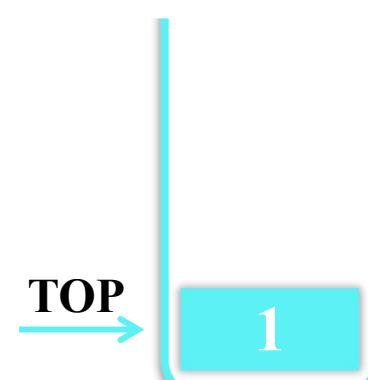
value(top)
 $<$
value(t)



t = Day2

$$\text{stock span} = [1, 2, 1]$$

value(top)
 $>$
value(t)

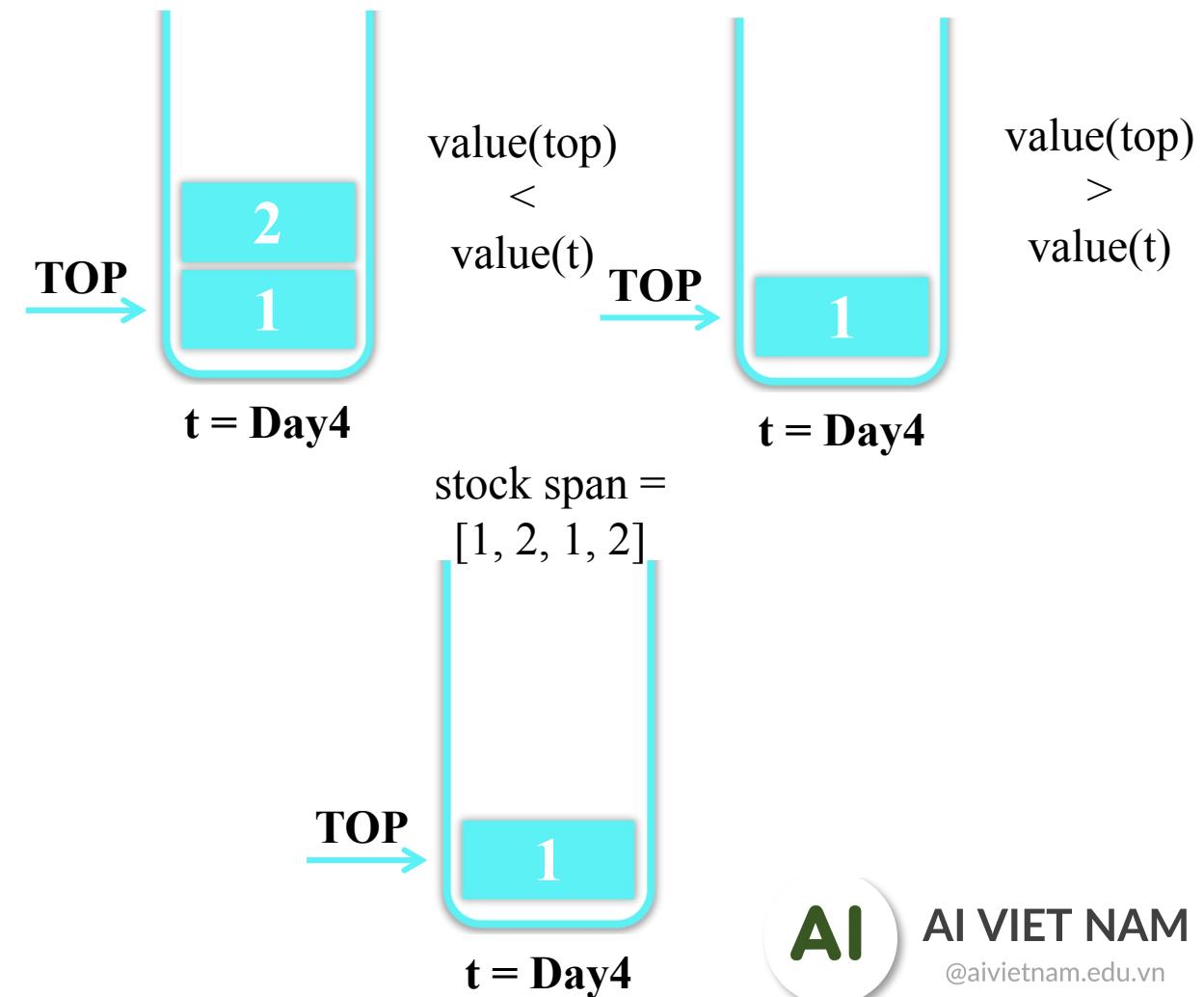
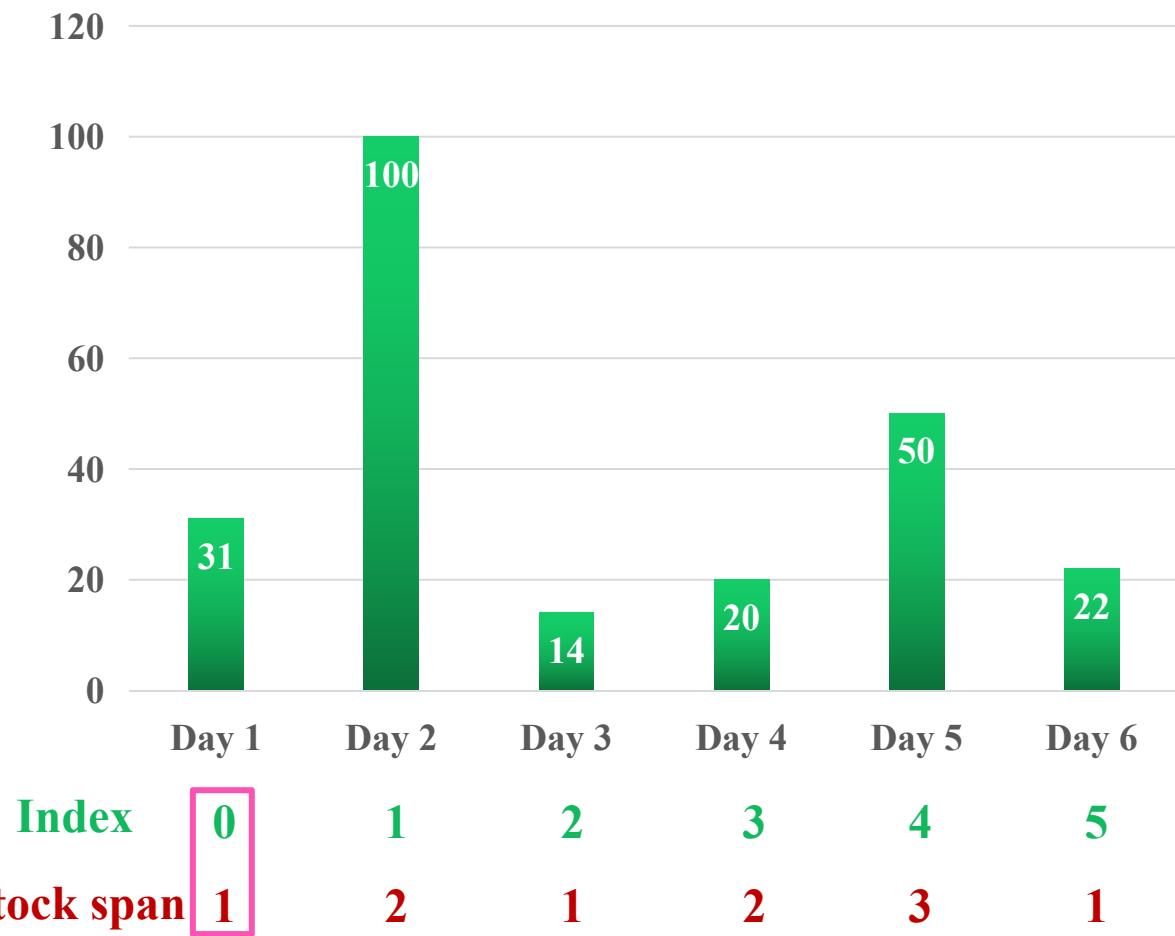


t = Day3

Exercise4: Application of Stack - Stock Span Problem



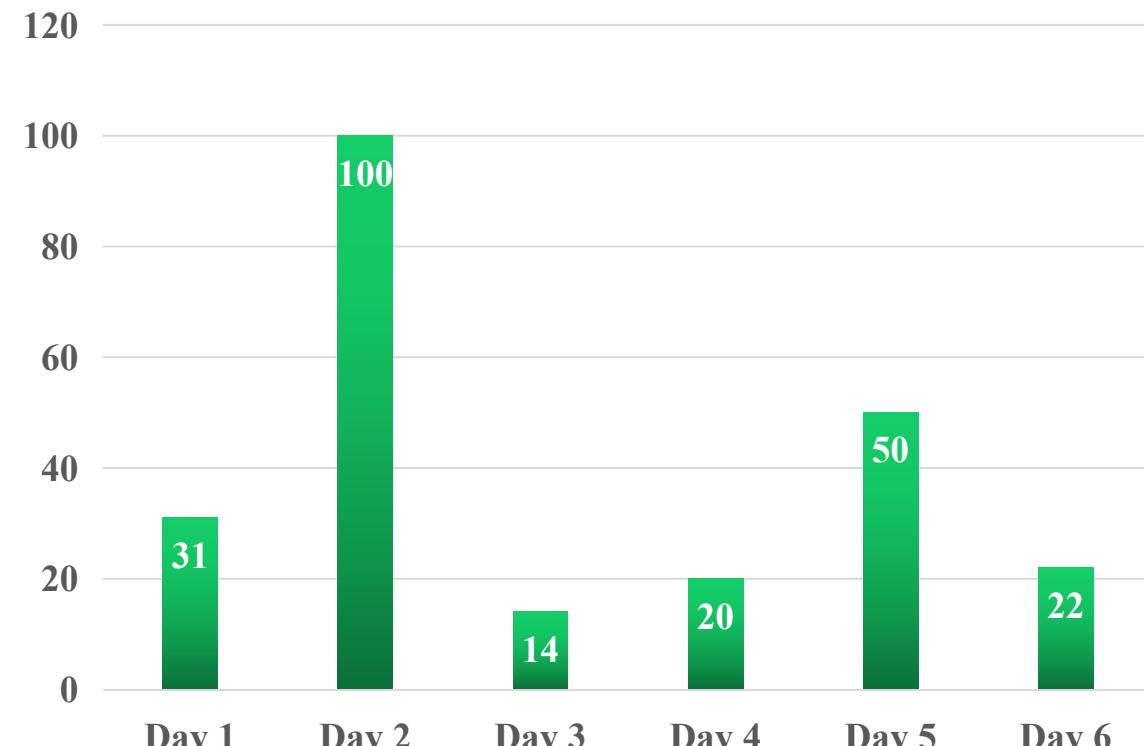
Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



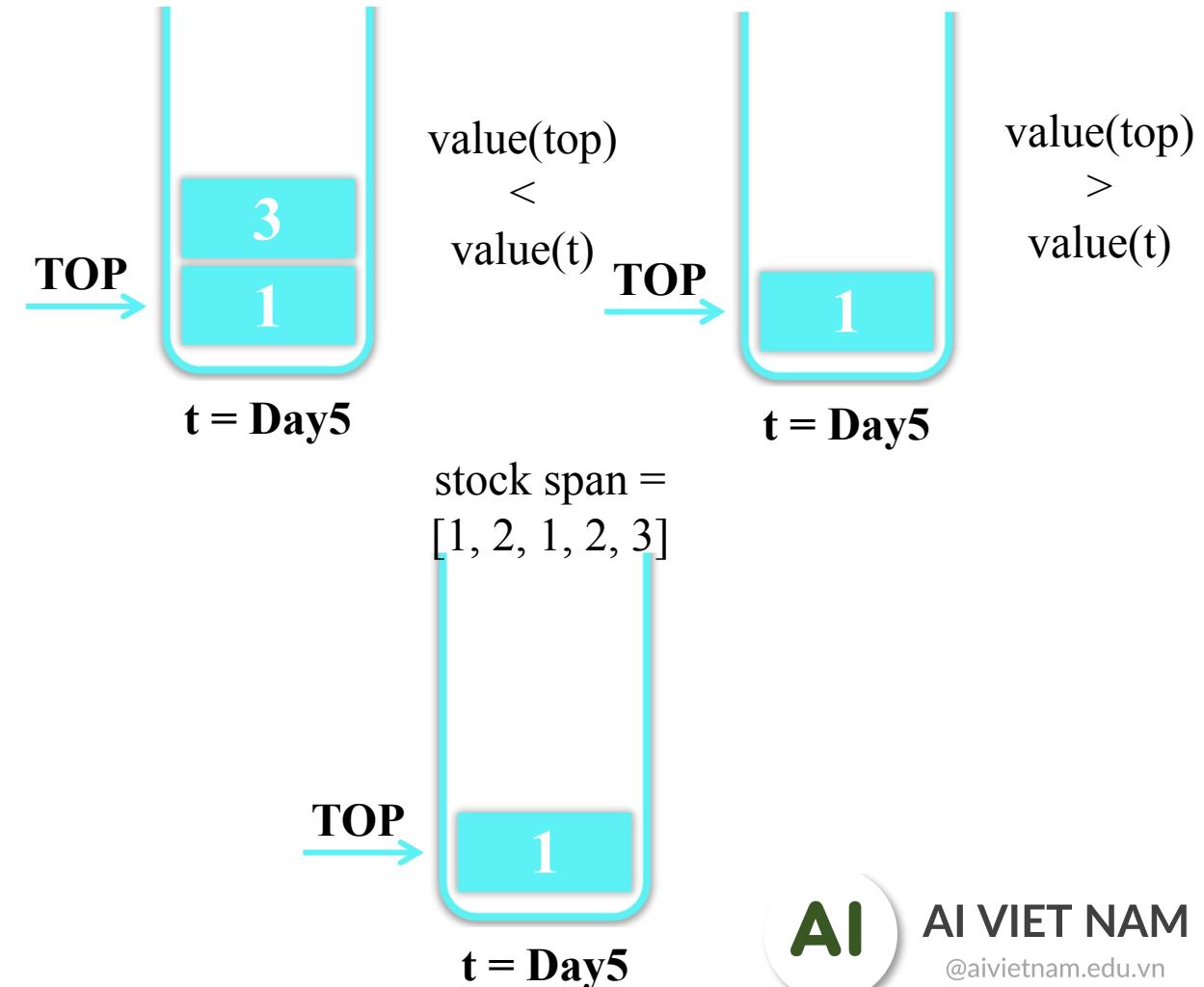
Exercise4: Application of Stack - Stock Span Problem



Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



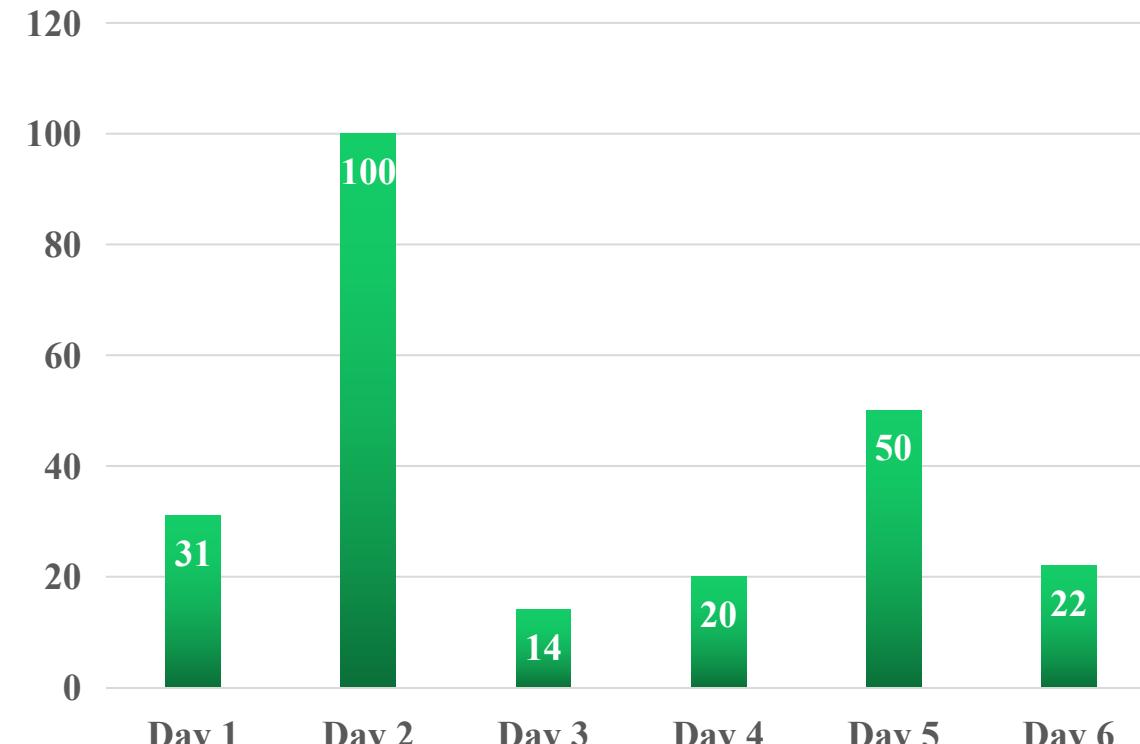
Index	Stock span
0	1
1	2
2	1
3	2
4	3
5	1



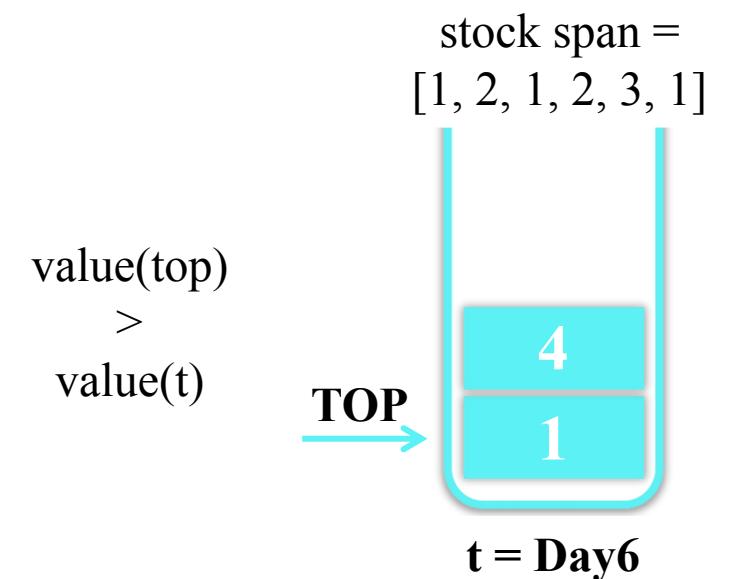
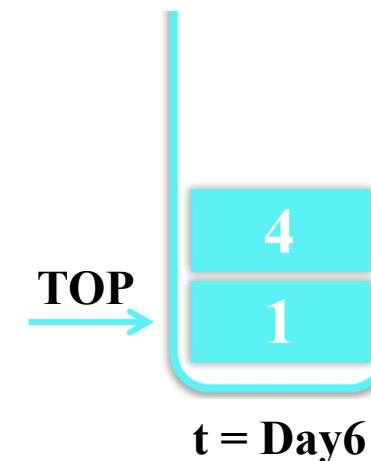
Exercise4: Application of Stack - Stock Span Problem



Stock Span: của một ngày hiện tại chính là số lượng ngày liên tiếp tối đa bắt đầu từ ngày hiện tại và cho đến những ngày trước trong quá khứ có **giá cổ phiếu \leq giá cổ phiếu của ngày hiện tại**



Index	0	1	2	3	4	5
Stock span	1	2	1	2	3	1



Exercise4: Application of Stack - Stock Span Problem

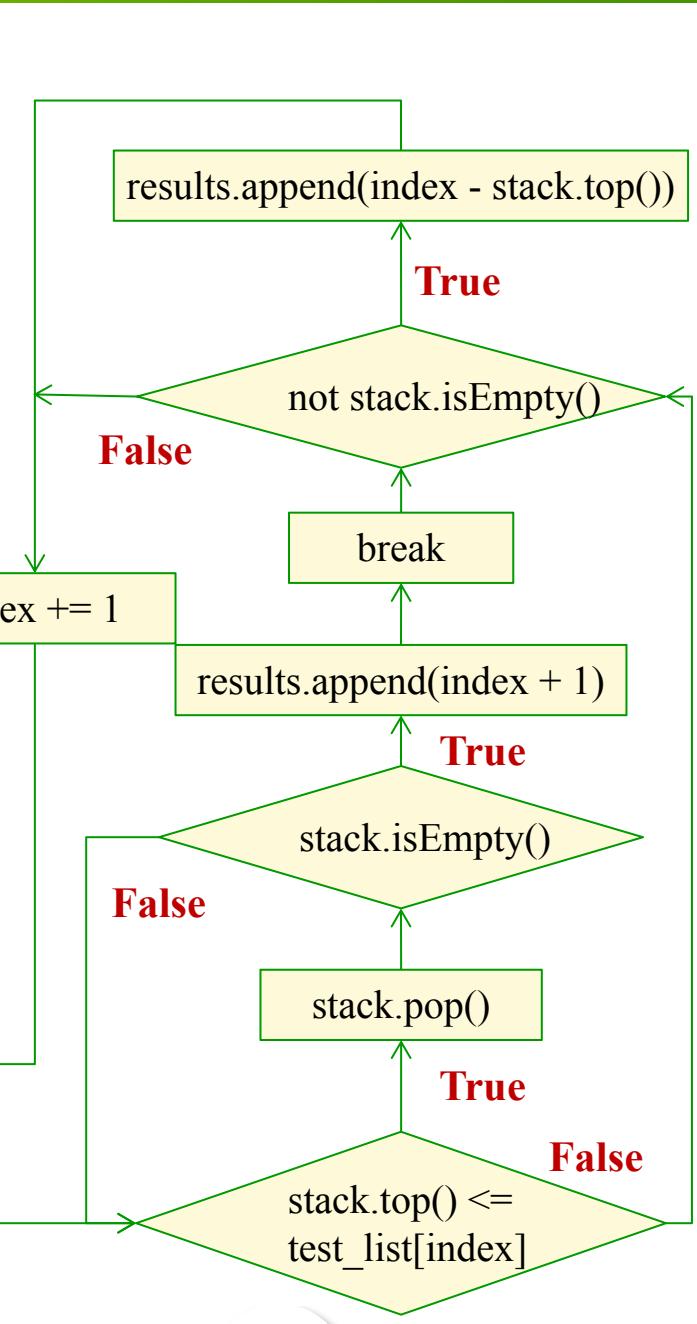
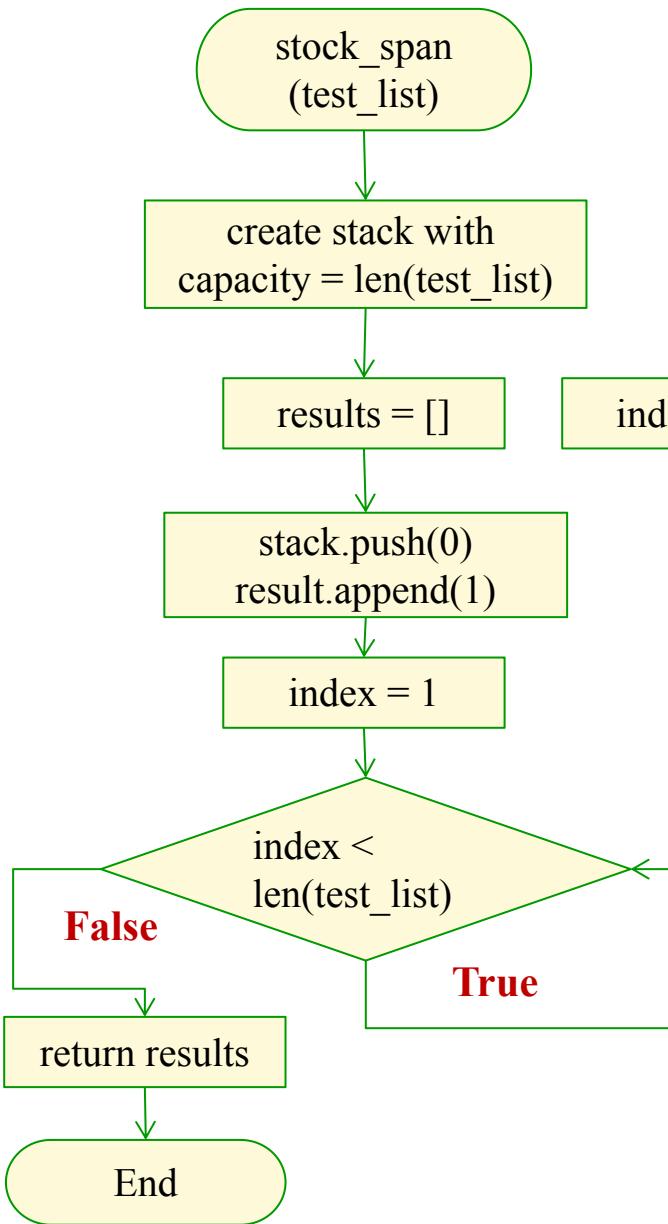
```
FUNCTION stock_span(test_list)
    create stack with capacity = len(test_list)
    results = list()
    stack.push(0)
    result.append(1)

    FOR index start at 1 TO len(test_list) - 1
        WHILE stack.top() <= test_list[index]
            stack.pop()
            IF stack.isEmpty()
                results.append(index + 1)
                break
            ENDIF
        ENDWHILE

        IF not stack.isEmpty()
            results.append(index - stack.top())
        ENDIF
    ENDFOR

    RETURN results

ENDFUNCTION
```





- **Exercise4: Application of Stack - Stock Span Problem**
 - Tính số ngày tối đa liên tiếp từ ngày hiện tại cho đến các ngày trước trong quá khứ có giá cổ phiếu thấp hơn hoặc bằng giá cổ phiếu hiện tại
- **Exercise5: Application of Queue - Generate Binary number from 1 to N**
 - Tạo ra chuỗi binary từ 1 đến N và đưa kết quả vào một dictionary
- **Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA**
 - Tìm và trả về một list các chuỗi có 10 ký tự được lặp lại hơn một lần trong chuỗi DNA cho trước

Exercise5: Application of Queue - Generate Binary number from 1 to N



Cho trước một range với N số nguyên dương (từ 1 đến N). Viết function biến đổi các số thập phân trong range sang dạng binary string và trả về một dictionary với key là số thập phân và value là binary dạng string

- **Input:** N là số nguyên dương thể hiện range từ 1 đến N
- **Output:** một dictionary với key là số thập phân và value là binary dạng string tương ứng
- **Khuyến khích** các bạn sử dụng queue, và nếu có thể thì sử dụng queue của các bạn đã thực hiện ở bài 3

```
1 # Examples
2 generateBinaryString(8)
3 >> {1: '1',
4 2: '10',
5 3: '11',
6 4: '100',
7 5: '101',
8 6: '110',
9 7: '111',
10 8: '1000'}
```

Exercise5: Application of Queue - Generate Binary number from 1 to N



Binary Number System

110100

A Binary Number is made up of only **0s** and **1s**.

Count in Decimal?

- | | |
|-----|---|
| 0 | Start at 0 |
| ... | Count 1,2,3,4,5,6,7,8, and then... |
| 9 | This is the last digit in Decimal |
| 10 | So we start back at 0 again, but add 1 on the left |

Binary	
0	Start at 0
1	Then 1
???	no symbol for 2 ... what do we do?

Binary	
0	Start at 0
1	Then 1
10	start back at 0 again, but add 1 on the left
11	1 more
???	what ... ?

What happens in Decimal?

- | | |
|-----|---|
| 99 | When we run out of digits |
| 100 | Start back at 0 again, but add 1 on the left |



Exercise5: Application of Queue - Generate Binary number from 1 to N



Binary Number System

A Binary Number is made up of only **0s** and **1s**.

Binary	
0	Start at 0
1	Then 1
10	start back at 0 again, but add 1 on the left
11	1 more
100	start back at 0 again, and add one to the number on the left... ... but that number is already at 1 so it also goes back to 0 and 1 is added to the next position on the left
101	
110	
111	
1000	Start back at 0 again (for all 3 digits), add 1 on the left
1001	And so on!

Exercise5: Application of Queue - Generate Binary number from 1 to N



```
1 # Examples
2 generateBinaryString(8)
3 >> {1: '1',
4 2: '10',
5 3: '11',
6 4: '100',
7 5: '101',
8 6: '110',
9 7: '111',
10 8: '1000'}
```

```
12 generateBinaryString(16)
13 >> {1: '1',
14 2: '10',
15 3: '11',
16 4: '100',
17 5: '101',
18 6: '110',
19 7: '111',
20 8: '1000',
21 9: '1001',
22 10: '1010',
23 11: '1011',
24 12: '1100',
25 13: '1101',
26 14: '1110',
27 15: '1111',
28 16: '10000'}
```

Exercise4: Application of Stack - Stock Span Problem



• Hiểu yêu cầu đề bài

Viết 1 Function

Input N (1 số nguyên dương)

Tạo thành range 1 đến N

Biến đổi các số trong range
từ decimal sang
binary dạng string

Output dictionary với
key là decimal
value là binary string

Khuyến khích sử dụng queue

Cho trước một range với **N số nguyên dương (từ 1 đến N)**. Viết **function biến đổi** các số thập phân trong range sang dạng **binary string** và trả về một **dictionary** với **key là số thập phân** và **value là binary dạng string**

- **Input:** N là số nguyên dương thể hiện range từ 1 đến N
- **Output:** một dictionary với key là số thập phân và value là binary dạng string tương ứng
- **Khuyến khích các bạn sử dụng queue, và nếu có thể thì sử dụng queue của các bạn đã thực hiện ở bài 3**

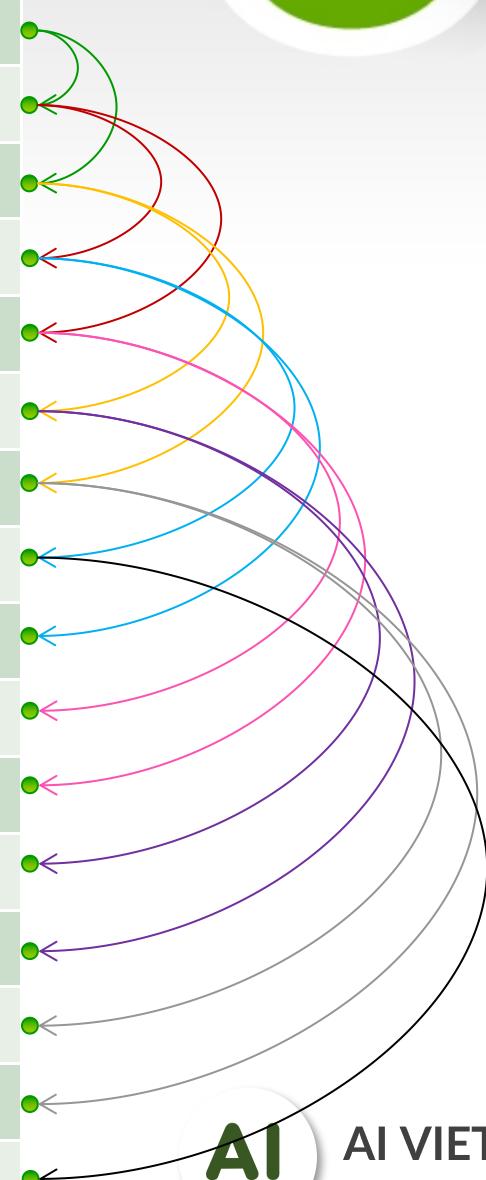
Exercise5: Application of Queue - Generate Binary number from 1 to N



Binary Number System

```
12 generateBinaryString(16)
13 >> {1: '1',
14 2: '10',
15 3: '11',
16 4: '100',
17 5: '101',
18 6: '110',
19 7: '111',
20 8: '1000',
21 9: '1001',
22 10: '1010',
23 11: '1011',
24 12: '1100',
25 13: '1101',
26 14: '1110',
27 15: '1111',
28 16: '10000'}
```

Decimal	Binary
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000



Exercise4: Application of Stack - Stock Span Problem



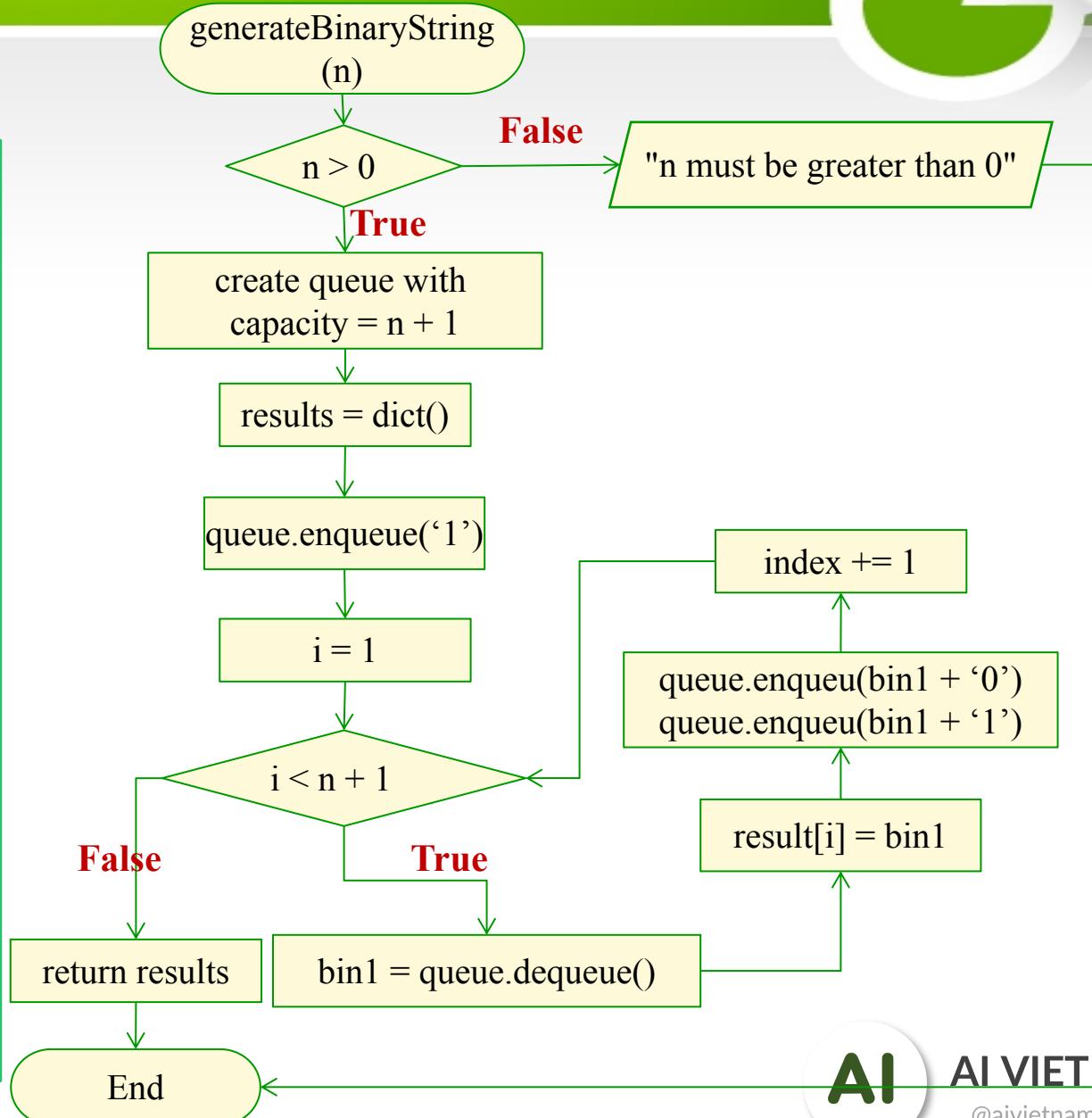
```
FUNCTION generateBinaryString(n)
IF n > 0
    create queue with capacity = n+1
    results = dict()
    queue.enqueue('1')

    FOR i start at 1 TO n
        bin1 = queue.dequeue()
        results[i] = bin1

        queue.enqueue(bin1 + '0')
        queue.enqueue(bin1 + '1')
    ENDFOR

    ELSE
        PRINT "n must be greater than 0"
        RETURN
    ENDIF

    RETURN results
ENDFUNCTION
```





- **Exercise4: Application of Stack - Stock Span Problem**
 - Tính số ngày tối đa liên tiếp từ ngày hiện tại cho đến các ngày trước trong quá khứ có giá cổ phiếu thấp hơn hoặc bằng giá cổ phiếu hiện tại
- **Exercise5: Application of Queue - Generate Binary number from 1 to N**
 - Tạo ra chuỗi binary từ 1 đến N và đưa kết quả vào một dictionary
- **Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA**
 - Tìm và trả về một list các chuỗi có 10 ký tự được lặp lại hơn một lần trong chuỗi DNA cho trước

Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA



Cho một chuỗi DNA, được tạo bởi các nucleotides 'A', 'C', 'G' và 'T'. Khi nghiên cứu về DNA cần nhận dạng các chuỗi được lặp lại trong DNA. Bài toán cho một chuỗi s là DNA, hãy trả về các chuỗi có 10 chữ cái được lặp lại nhiều hơn một lần dựa vào s (Thứ tự trả về không quan trọng).

- **Input:** s là chuỗi DNA dạng string chỉ có các ký tự 'A', 'C', 'G' và 'T'
- **Output:** list các chuỗi có 10 ký tự được lặp lại hơn một lần
- **Hint:** các bạn có thể sử dụng deque

```
1 s = "TTTTTTTTTTTTTTTT"
2 findRepeatedDnaSequences(s)
3 >> ['TTTTTTTTTT']
4
5 s = "TTTTGGGGTTTTGGGGGTTTTAAACCC"
6 findRepeatedDnaSequences(s)
7 >> ['TTTTGGGG', 'GGGGGTTTT']
8
9 s = "TTTTGGGGTTTTGGGG"
10 findRepeatedDnaSequences(s)
11 >> ['TTTTGGGG']
```

Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA



• Hiểu yêu cầu đề bài

Viết 1 Function

Input một chuỗi DNA s

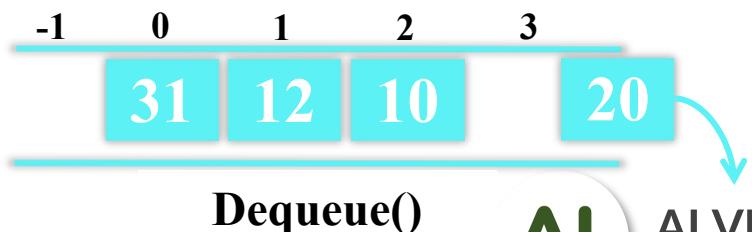
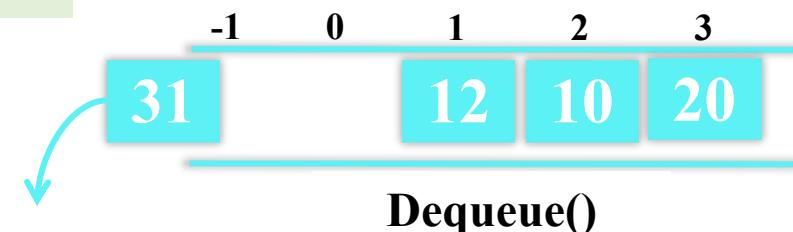
Tìm các chuỗi con có 10
chữ cái lặp lại hơn 1 lần

Output list các chuỗi con
lặp lại hơn 1 lần

Hint: có thể dùng deque

Cho một **chuỗi DNA**, được tạo bởi các nucleotides '**A**', '**C**', '**G**' và '**T**'. Khi nghiên cứu về DNA cần phải nhận dạng các chuỗi được lặp lại trong DNA. Bài toán **cho một chuỗi s là DNA**, hãy trả về các chuỗi có 10 chữ cái được **lặp lại nhiều hơn một lần** dựa vào s (Thứ tự trả về không quan trọng).

- **Input:** s là chuỗi DNA dạng string chỉ có các ký tự 'A', 'C', 'G' và 'T'
- **Output:** list các chuỗi có 10 ký tự được lặp lại hơn một lần
- **Hint:** các bạn có thể sử dụng deque



Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA



- TTTTTGGGGGGTTTTGGGGGGGTTTAAACCC
 - TTTTTGGGGGGTTTTGGGGGGGTTTAAACCC
 - TT~~TTTGGGGGGT~~TTTGGGGGGGTTTAAACCC
 - TTTTTGGGGGGTTTTGGGGGGGTTTAAACCC
 - TTTTT~~TGGGGGGT~~TTTGGGGGGGTTTAAACCC
 - TTTTT~~GGGGGGT~~TTTGGGGGGGTTTAAACCC
 - TTTTTGG~~GGGGT~~TTTGGGGGGGTTTAAACCC
 - TTTTTGGGG~~GGT~~TTTGGGGGGGTTTAAACCC
 - TTTTTGGGGGG~~T~~TTTGGGGGGGTTTAAACCC
 - TTTTTGGGGGGGTTTGGGGGGGTTTAAACCC

⋮ ⋮ ⋮

 - TTTTTGGGGGGGTTTTGGGGGGGTTTAAACCC

Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA

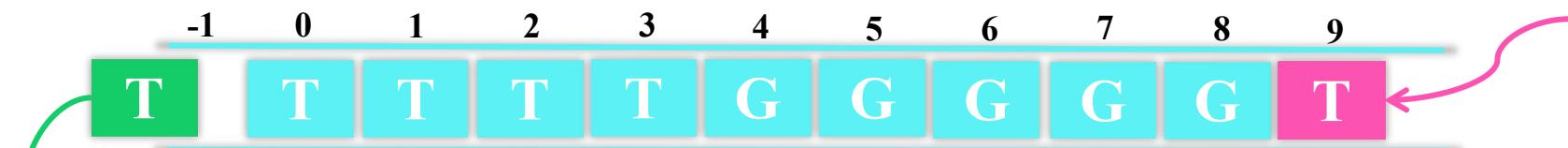
Dictionary
- key là chuỗi con
- value là số lần xuất hiện

- TTTTGGGGGGTTTTGGGGGGTTTTAAACCC



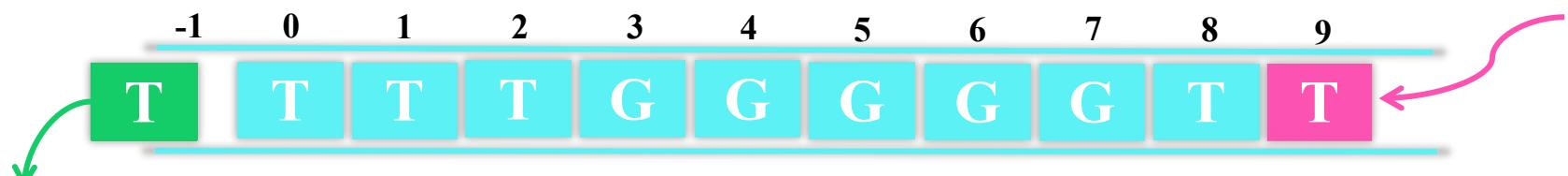
Dictionary
- "TTTTGGGGGG" : 1

- TTTTGGGGGGTTTTGGGGGGTTTTAAACCC



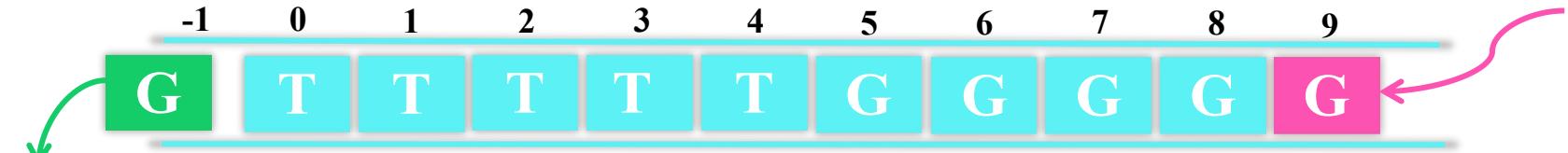
Dictionary
- "TTTTGGGGGG" : 1
- "TTTGGGGGT" : 1

- TT~~TTTGGGGGG~~T~~TTTGGGGGG~~TTTTAAACCC



Dictionary
- "TTTTGGGGGG" : 1
- "TTTGGGGGT" : 1
- "TTTGGGGGTT" : 1

- TTTTGGGGGG~~TTTTTGGGGGG~~TTTTAAACCC



Dictionary
- "TTTTGGGGGG" : 2
- "TTTGGGGGT" : 1
- "TTTGGGGGTT" : 1
- ...

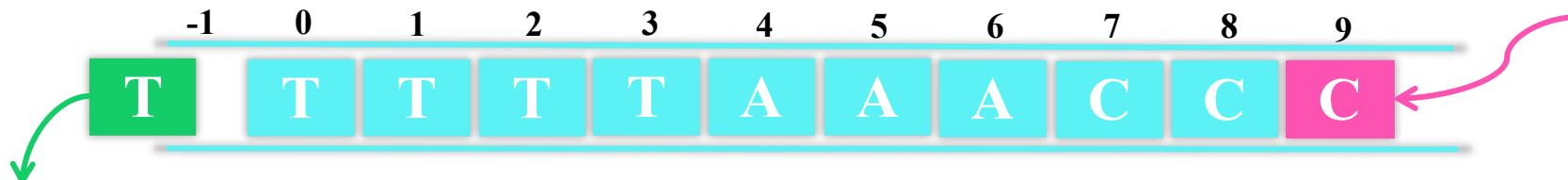
Optional Exercise: Tìm chuỗi con lặp trong một chuỗi DNA



Dictionary

- key là chuỗi con (10 chars)
- value là số lần xuất hiện

• TTTTTGGGGGTTTTGGGGGGTTTTTAAACCC



Final Dictionary

- "TTTTTGGGGG" : 2
- "TTTTGGGGGT" : 1
- "TTTGGGGGTT" : 1
- ...
- "TTTTAAACCC" : 1

Final Dictionary

- "TTTTTGGGGG" : 2
- "TTTTGGGGGT" : 1
- "TTTGGGGGTT" : 1
- ...
- "TTTTAAACCC" : 1

- Tạo một list results

- Loop trong Final Dictionary
- Nếu item nào có value > 1
- thì append key vào list results
- Loop xong thì return results

