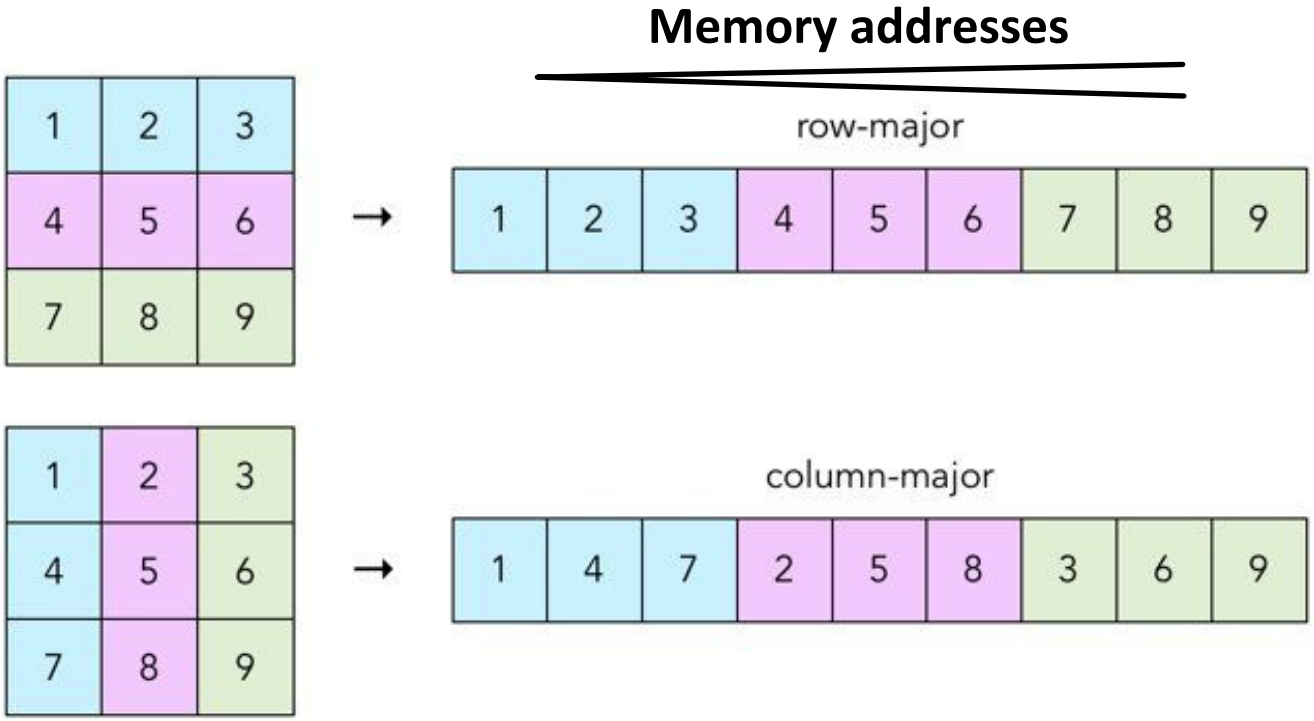
Array

- A **scalar variable** stores the value of a datum.
- A **1D array** variable stores the value of a plural of data *of the same type*. In other words, an array variable is a set of a plural of (scalar) variables of the same type.
- With C++ programming language, an array is equivalent to a pointer to the first element of the array: int myArray[10];
- In C++, **multidimensional arrays are just an array of arrays**: int my2dArray[12][14].

Row-major vs. Column-major

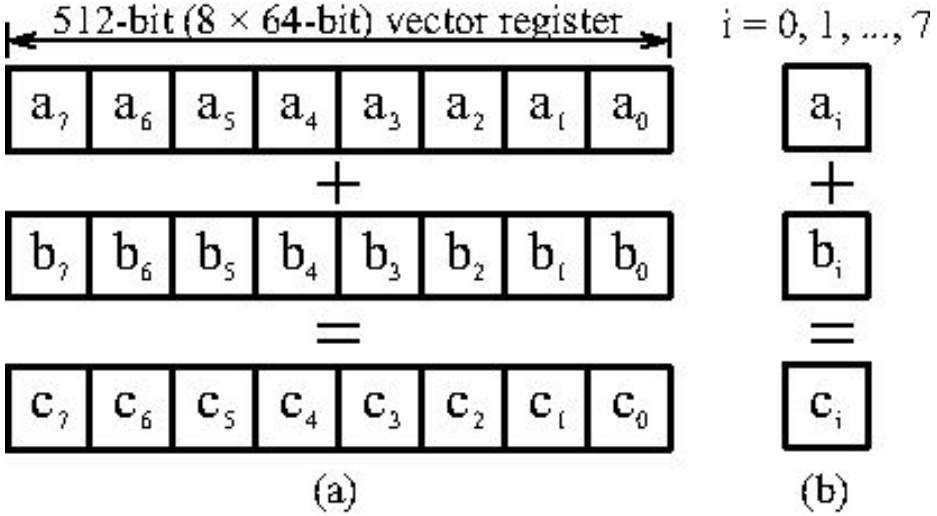
- Traversing a row of an array is faster than traversing a column, because C++ uses row-major order for array elements.



https://craftofcoding.wordpress.com/2017/02/03/column-major-vs-row-major-arrays-does-it-matter/

Biginteger

- RAM size limit.
- Python Biginteger API.
- One array for one digit.
- Epsilon of long float.



SUM

```
c[i] = a[i] + b[i] + carry;
carry = c[i] / 10;
c[i] %= 10;
```

運算時不用立即進位，可以後來再去一口氣進位

Minus

```
c[i] = a[i] - b[i] - borrow;
if(c[i] < 0){
        borrow = 1;
        c[i] += 10;
}
else
        borrow = 0;
```

運算時若小於0
則需考慮借位

Multiply

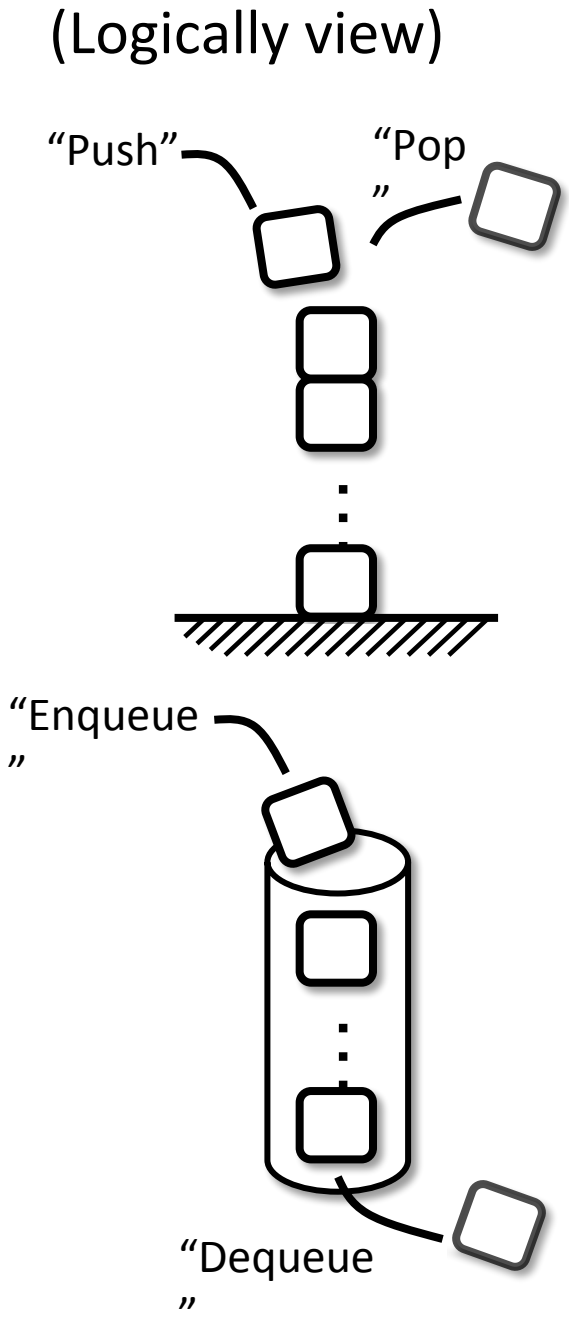
```
for(i = 0; i < 100; ++i ){
        if(a[i]==0) continue;
        for(j = 0; j < MAX; ++j)
                c[i+j] += a[i] * b[i];
}

for(i = 0; i < MAX; ++i){
        carry = c[i] / 10;
        c[i] %= 10;
}
```

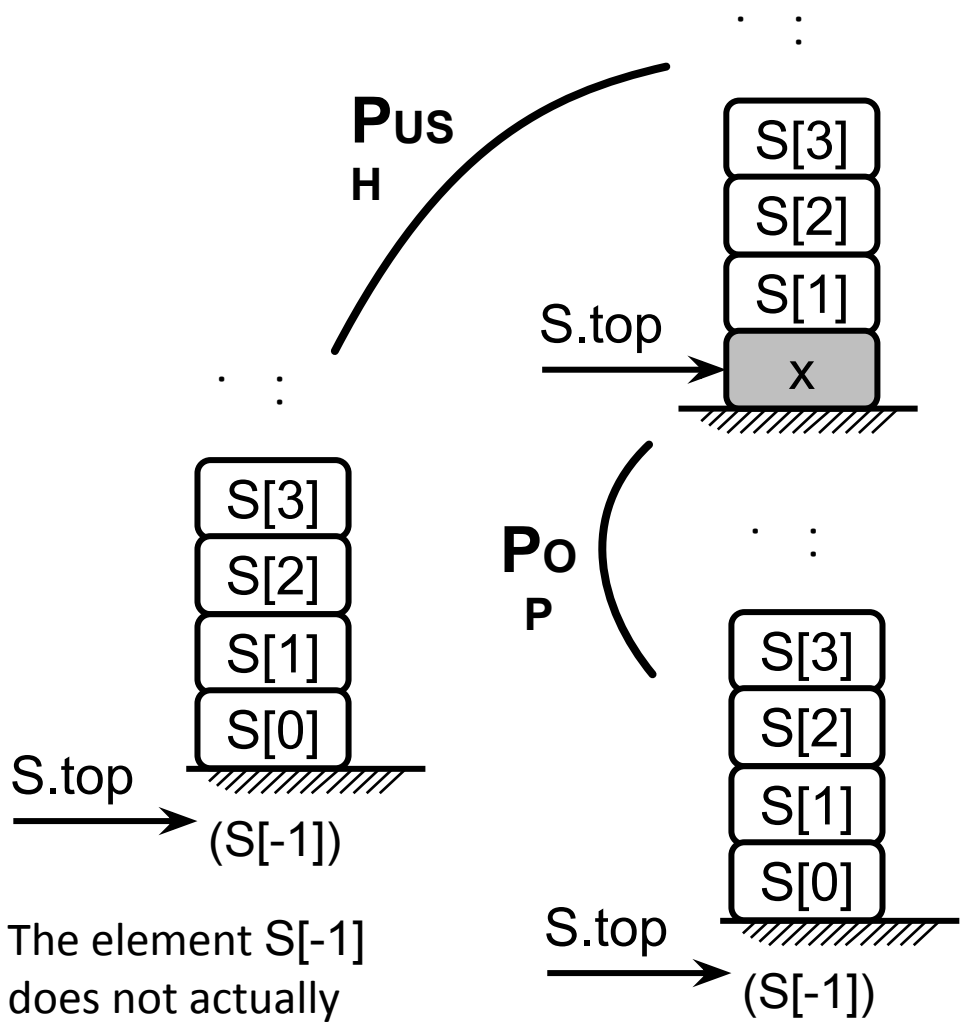
運算時不用立即進位，可以後來再去一口氣進位

Stack vs. Queue

- Conceptual views:
  - In a **stack**, the element deleted from the set is the one most recently inserted
    → A stack uses a **last-in first-out (LIFO) policy**.
  - In a **queue**, the element deleted is always the one that has been in the set for the longest time
    → A queue uses a **first-in first-out (FIFO) policy**.



Use an Array to Implement a Stack



STACK-EMPTY(S)

```
if S.top == -1
    return true;
else return false;
```

PUSH(S, x)

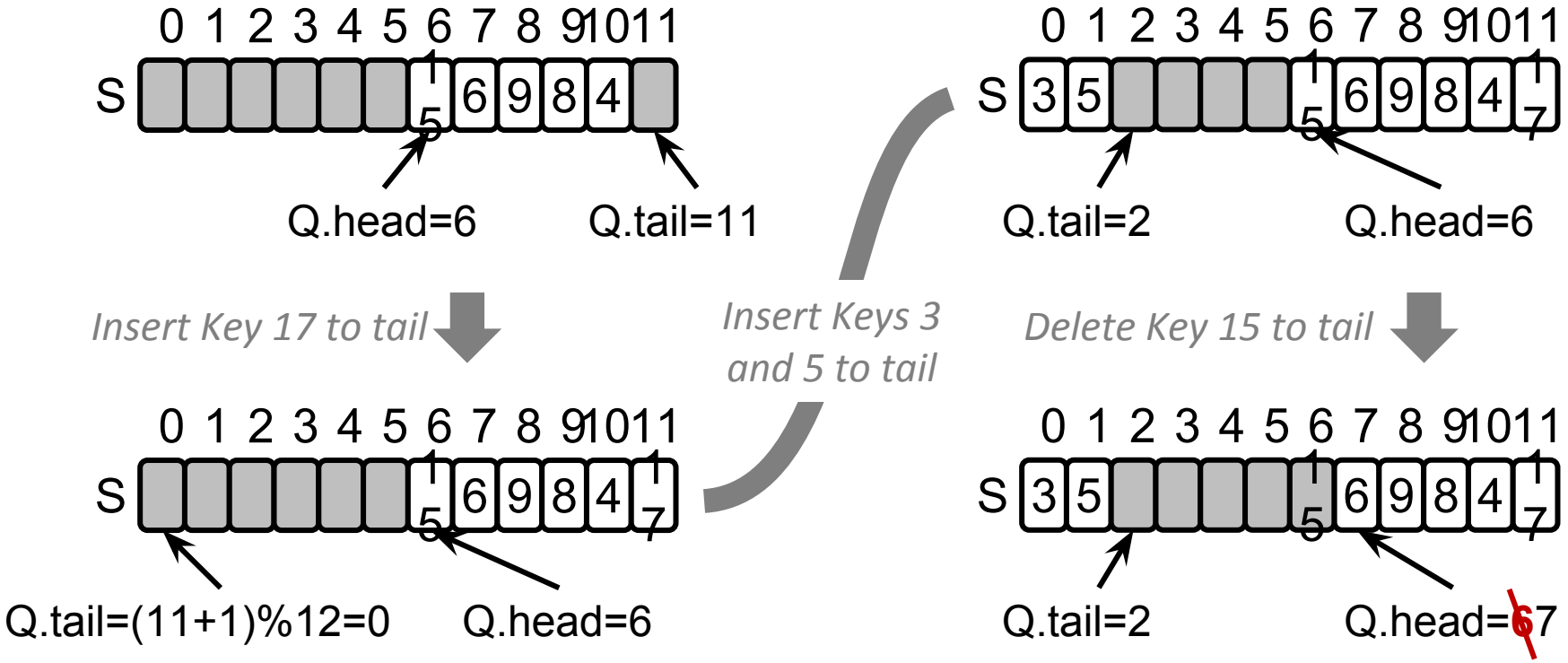
```
S.top++;
S[S.top] = x;
```

We assume that the array doe not overflow, but this is not true…

POP(S)

```
if Stack-Empty(S)
    error underflow;
else
    S.top--;
    return S[S.top+1];
```

Use an Array to Implement a Queue



- Exercise: Complete the following functions:
  - **Queue-Empty** (check if the queue is empty)
  - **Enqueue** (insert an item to the queue)
  - **Dequeue** (delete an item from the queue)

Solutions (with Some Problems)

```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
1  x = Q[Q.head]
2  if Q.head == Q.length
3        Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

Underflow vs. Overflow

- If a queue or stack is implemented using a limited-size array,
  - **Overflows** occur when we insert an element (by enqueue or push) to a full queue or stack.
  - **Underflows** may occur if we try to remove an element (by dequeue or pop) from an empty queue or stack.

String

- more: Python3字符串各种内置函数详解

Defining a Class

- A *class* is a special data type which defines how to build a certain kind of object.
- The *class* also stores some data items that are shared by all the instances of this class
- *Instances* are objects that are created which follow the definition given inside of the class
- Python doesn't use separate class interface definitions as in some languages
- You just define the class and then use it

Methods in Classes

- Define a *method* in a *class* by including function definitions within the scope of the class block
- There must be a special first argument ***`self`*** in *all* of method definitions which gets bound to the calling instance
- There is usually a special method called ***`__init__`*** in most classes
- We'll talk about both later…

A simple class def: *student*

```
class student:
 """A class representing a
 student """
 def __init__(self,n,a):
      self.full_name = n
      self.age = a
 def get_age(self):
      return self.age
```

Creating and Deleting Instances

Instantiating Objects

- There is no “new” keyword as in Java.
- Just use the class name with ( ) notation and assign the result to a variable
- `__init__` serves as a constructor for the class. Usually does some initialization work
- The arguments passed to the class name are given to its `__init__()` method
- So, the __init__ method for student is passed “Bob” and 21 and the new class instance is bound to b:

```
b = student(“Bob”, 21)
```

Constructor: __init__

- An `__init__` method can take any number of arguments.
- Like other functions or methods, the arguments can be defined with default values, making them optional to the caller.

- However, the first argument `self` in the definition of __init__ is special…

Self

- The first argument of every method is a reference to the current instance of the class
- By convention, we name this argument ***self***
- In `__init__`, *self* refers to the object currently being created; so, in other class methods, it refers to the instance whose method was called
- Similar to the keyword *this* in Java or C++
- But Python uses *self* more often than Java uses *this*

Self

- Although you must specify *self* explicitly when *defining* the method, you don't include it when *calling* the method.
- Python passes it for you automatically

Defining a method:

(this code inside a class definition.)

```
def set_age(self, num):
  self.age = num
```

Calling a method:

```
>>> x.set_age(23)
```

Deleting instances: No Need to “free”

- When you are done with an object, you don’t have to delete or free it explicitly.
- Python has automatic garbage collection.
- Python will automatically detect when all of the references to a piece of memory have gone out of scope. Automatically frees that memory.
- Generally works well, few memory leaks
- There’s also no “destructor” method for classes

Access to Attributes and Methods

Definition of student

```
class student:
 """A class representing a student
 """
 def __init__(self,n,a):
     self.full_name = n
     self.age = a
 def get_age(self):
     return self.age
```

Traditional Syntax for Access

```
>>> f = student("Bob Smith", 23)

>>> f.full_name # Access attribute
"Bob Smith"

>>> f.get_age() # Access a method
23
```

Accessing unknown members

- Problem: Occasionally the name of an attribute or method of a class is only given at run time…
- Solution:

```
getattr(object_instance, string)
```

- **`string`** is a string which contains the name of an attribute or method of a class
- **`getattr(object_instance, string)`** returns a reference to that attribute or method