

Introduction to Programming PYTHON

Chapter 4 – Sequences, Sets, Arrays and Dictionary

Presenter: **Dr. Nguyen Dinh Long**

Email: dinhlonghcmut@gmail.com

Google-site: <https://sites.google.com/view/long-dinh-nguyen>

Oct. 2022

Programming

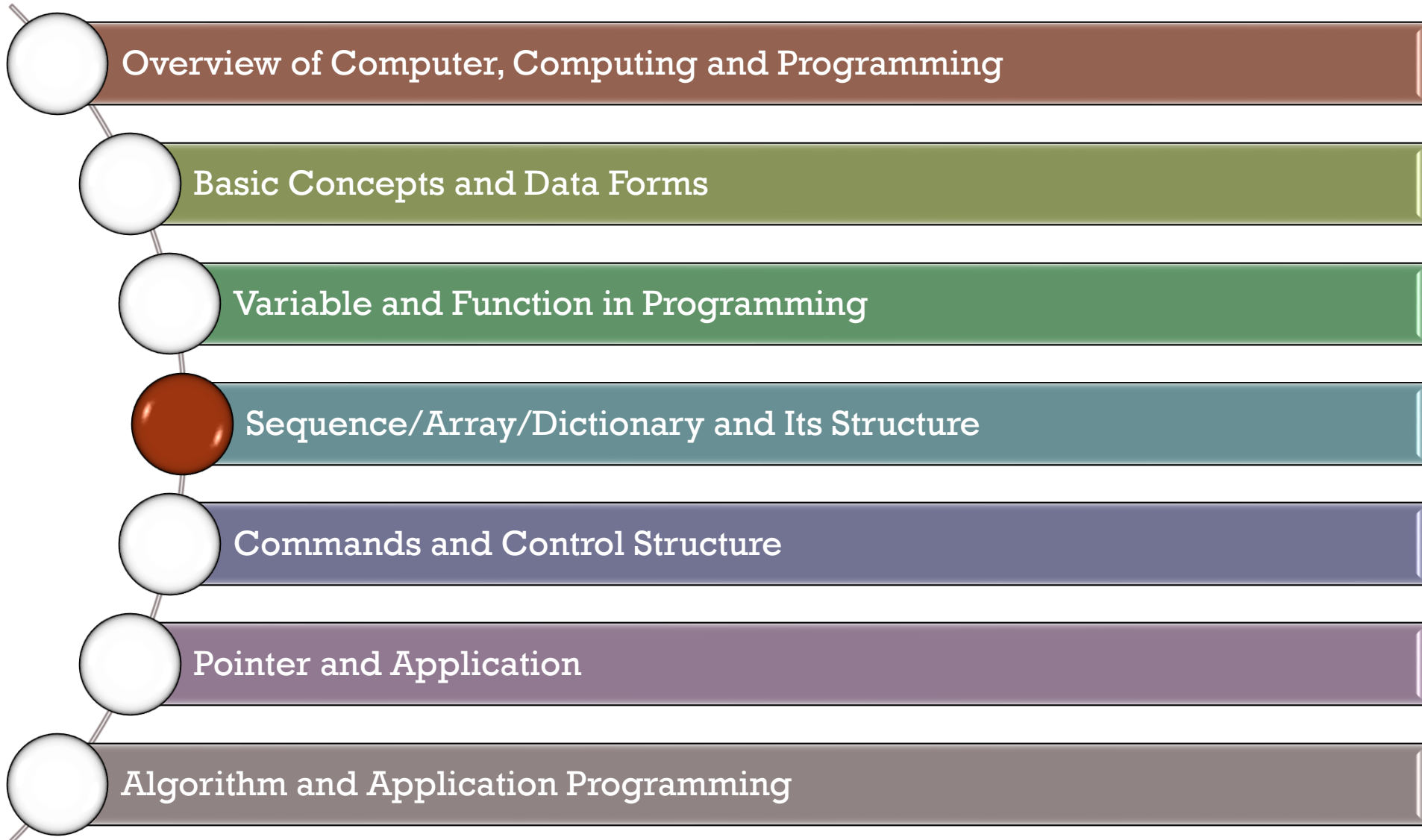
Hamilton was a self-taught programmer, working in the US in the 1960's. Owing to the success of her previous work, Hamilton was the first programmer to be hired for the Apollo project. She became the Director of Software Engineering at the MIT Instrumentation lab. Her lab developed the on-board flight software for NASA's Apollo space project, which took humankind to the moon.

The achievement was a monumental task at a time when computer technology was in its infancy: The astronauts had access to only 72 kilobytes of computer memory (a 256-gigabyte cell phone today carries almost a million times more storage space). Programmers had to use paper punch cards to feed information into room-sized computers with no screen interface.

Margaret Hamilton, NASA's lead software engineer for the Apollo, stands next to the code she wrote by hand that took humanity to the moon in 1969.



Outline



References

Main:

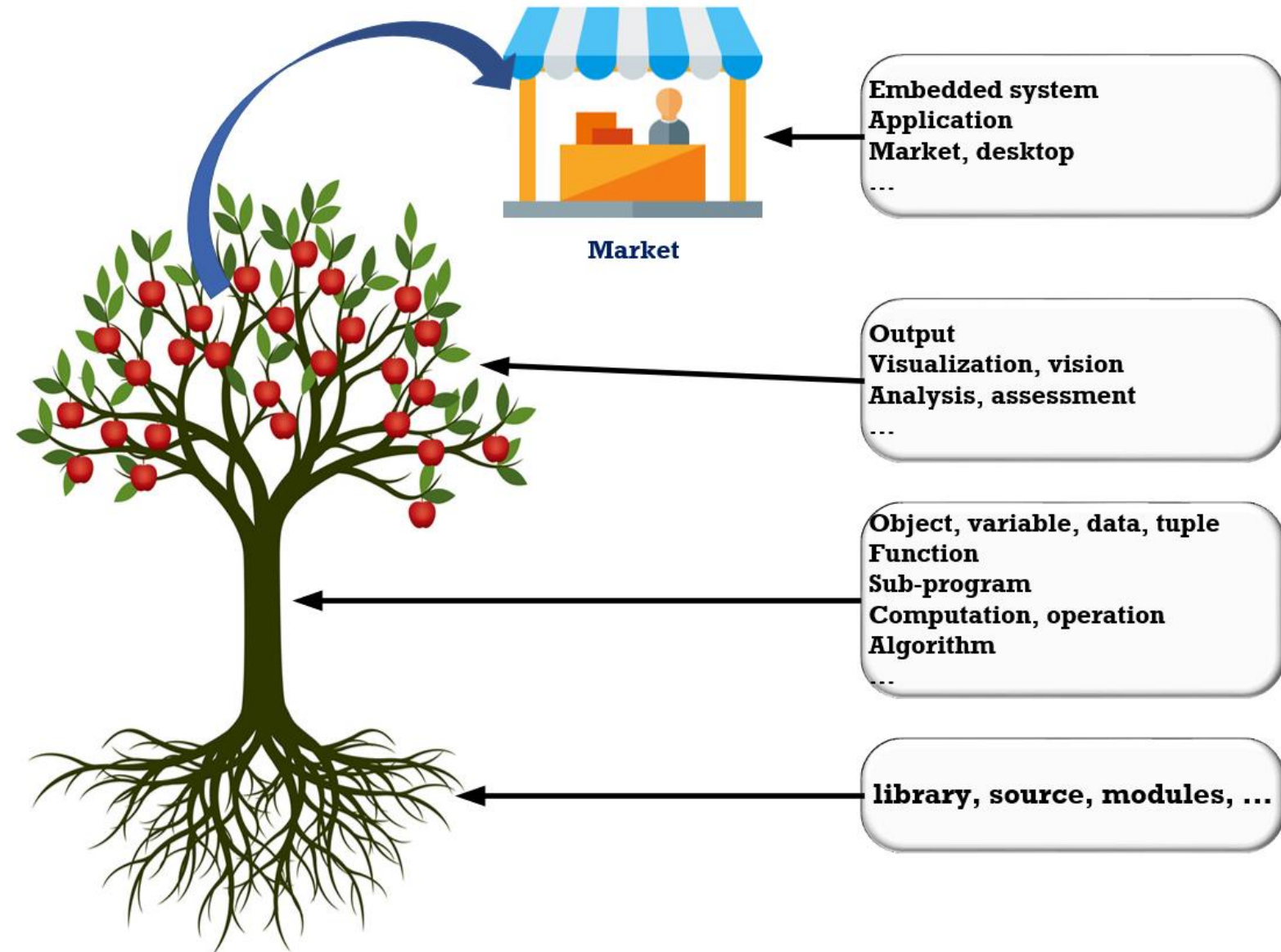
- Maurizio Gabbrielli and Simone Martini, 2010. *Programming Languages: Principles and Paradigms*, Springer.
- Cao Hoàng Trữ, 2004. *Ngôn ngữ lập trình- Các nguyên lý và mô hình*, Nhà xuất bản Đại học Quốc gia Tp. Hồ Chí Minh

More:

- Wes McKinney, 2013. *Python for Data Analysis*, O'Reilly Media.
- Guido van Rossum, Fred L. Drake, Jr., 2012. *The Python Library Reference*, Release 3.2.3.
- Slides here are collected and modified from several sources in Universities and Internet.

Computer programs

General structure:



Content of Chapter 4

1. Sequences: lists, strings and tuples, its structure
2. Sequences examples
3. Arrays: vectors, lists, matrix, tensor, its structure
4. Arrays examples
1. Practice

Sequences and Structure

❏ Lots of data:

➤ Lists

➤ Strings

➤ Tuples

Rank ↕	NOC ↕	Gold ↕	Silver ↕	Bronze ↕	Total ↕
1	 Norway (NOR)	14	14	11	39
2	 Germany (GER)	14	10	7	31
3	 Canada (CAN)	11	8	10	29
4	 United States (USA)	9	8	6	23
5	 Netherlands (NED)	8	6	6	20
6	 Sweden (SWE)	7	6	1	14
7	 South Korea (KOR)	5	8	4	17
8	 Switzerland (SUI)	5	6	4	15
9	 France (FRA)	5	4	6	15
10	 Austria (AUT)	5	3	6	14
11	 Japan (JPN)	4	5	4	13
12	 Italy (ITA)	3	2	5	10
13	 Olympic Athletes from Russia (OAR)	2	6	9	17
14	 Czech Republic (CZE)	2	2	3	7
15	 Belarus (BLR)	2	1	0	3
16	 China (CHN)	1	6	2	9
17	 Slovakia (SVK)	1	2	0	3
18	 Finland (FIN)	1	1	4	6
19	 Great Britain (GBR)	1	0	4	5
20	 Poland (POL)	1	0	1	2
21	 Hungary (HUN)	1	0	0	1
	 Ukraine (UKR)	1	0	0	1
23	 Australia (AUS)	0	2	1	3
24	 Slovenia (SLO)	0	1	1	2
25	 Belgium (BEL)	0	1	0	1

Sequences and Structure

▣ Lots of data:

How can we store this much data in Python? We would need 4×26 variables ...

The solution is to store all values together in a **list**.

Australia	0	2	1
Austria	5	3	6
Belarus	2	1	0
Belgium	0	1	0
Canada	11	8	10
China	1	6	2
Czech Republic	2	2	3
Finland	1	1	4
France	5	4	6
Germany	14	10	7
Great Britain	1	0	4
Hungary	1	0	0
Italy	3	2	5
Japan	4	5	4
Netherlands	8	6	6
New Zealand	0	0	2
Norway	14	14	11
Poland	1	0	1
Russia	2	6	9
Slovakia	1	2	0
Slovenia	0	1	1
South Korea	5	8	4
Sweden	7	6	1
Switzerland	5	6	4
Ukraine	1	0	0
United States	9	8	6



Sequences and Structure

Sequence - Lists:

To create a list, enclose the values in square brackets:

```
>>> countries = [ "Australia", ... , "United States" ]  
>>> gold = [0, 5, 2, 0, 11, 1, 2, 1, 5, 14, 1, 1, 3, 4, 8,  
0, 14, 1, 2, 1, 0, 5, 7, 5, 1, 9]
```

A list is an object of type **list**.

We can access the elements of a list using an integer index.

The first element is at index **0**, the second at index **1**, and soon:

```
>>> countries[0]  
'Australia'  
>>> countries[21]  
'South Korea'  
>>> gold[21]  
5
```

Negative indices start at the end of the list:

```
>>> countries[-1]  
'United States'  
>>> countries[-5]  
'South Korea'
```

Sequences and Structure

❑ Sequence – Lists (properties):

The length of a list is given by `len`:

```
>>> len(countries)
26
```

The empty list is written `[]` and has length zero. Lists can contain a mixture of objects of any type:

```
>>> korea = [ 'Korea', 'KOR', 5, 8, 4 ]
```

```
>>> korea[1]
```

```
'KOR'
```

```
>>> korea[2]
```

```
5
```

```
>>> korea = [ "Korea", 'KOR', (5, 8, 4) ]
```

Sequences and Structure

❑ Sequence – Lists (mutable):

A list of noble gases:

```
>>> nobles = [ 'helium', 'none', 'argon', 'krypton',  
               'xenon' ]
```

Oops. Correct the typo:

```
>>> nobles[1] = "neon"  
>>> nobles  
['helium', 'neon', 'argon', 'krypton', 'xenon']
```

Oops oops. I forgot radon!

```
>>> nobles.append('radon')  
>>> nobles  
['helium', 'neon', 'argon', 'krypton', 'xenon',  
 'radon']
```

Sequences and Structure

❑ Sequence – Lists (Aliasing):

Reminder: An object can have more than one name. This is called **aliasing**. We have to be careful when working with mutable objects:

```
>>> list1 = ["A", "B", "C"]
>>> list2 = list1
>>> len(list1)
3
>>> list2.append("D")
>>> len(list1)
4
>>> list1[1] = "X"
>>> list2
['A', 'X', 'C', 'D']
>>> list1 is list2
True
```

```
>>> list1 = ["A", "B", "C"]
>>> list2 = ["A", "B", "C"]
>>> len(list1)
3
>>> list2.append("D")
>>> len(list1)
3
>>> list1[1] = "X"
>>> list2
['A', 'B', 'C', 'D']
>>> list1 is list2
False
```

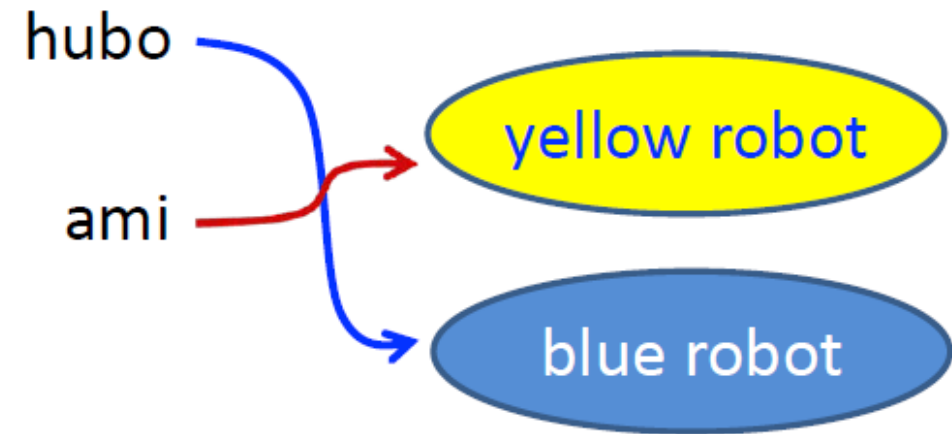
Sequences and Structure

❑ Sequence – Lists (Objects with two names):

To create a list, enclose the values in square brackets:

```
hubo = Robot("yellow")  
hubo.move()  
ami = hubo  
ami.turn_left()  
hubo.move()
```

```
hubo = Robot("blue")  
hubo.move()  
ami.turn_left()  
ami.move()
```



Sequences and Structure

❑ Sequence – Lists (Built-in functions):

`len` returns length of a list.

`sum` the sum of the elements.

`max` the largest element, `min` the smallest element:

```
>>> len(gold), sum(gold), max(gold), min(gold)
(26, 103, 14, 0)
```

```
>>> len(silver), sum(silver), max(silver)
(26, 102, 14)
```

```
>>> len(bronze), sum(bronze), max(bronze)
(26, 97, 11)
```

Sequences and Structure

❑ Sequence – Lists (Traversing):

A **for** loop looks at every element of a list:

```
for country in countries:  
    print(country)
```

We can get a range object from the **range** function as below:

```
>>> range(10)  
range(0, 10)  
>>> type(range(10))  
<class 'range'>  
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> list(range(10, 15))  
[10, 11, 12, 13, 14]
```

If we want to modify elements, we need the index:

```
>>> l = list(range(1, 11))  
>>> for i in range(len(l)):  
...     l[i] = l[i] ** 2  
>>> l  
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```


Sequences and Structure

❑ Sequence – Lists (Traversing):

Let's print out the total number of medals for each country:

```
>>> for i in range(len(countries)):
...     print(countries[i], gold[i]+silver[i]+bronze[i])
```

We can create a new list:

```
>>> totals = []
>>> for i in range(len(countries)):
...     medals = gold[i]+silver[i]+bronze[i]
...     totals.append( (medals, countries[i]) )
```

The list `totals` is now a list of tuples (**medals, country**).

```
>>> totals
[(3, 'Australia'), (14, 'Austria'), (3, 'Belarus'), ...,
(13, 'Japan'), (20, 'Netherlands'), ...,
(17, 'South Korea'), ..., (1, 'Ukraine'),
(23, 'United States')]
```

Sequences and Structure

❑ Sequence – Lists (Sorting):

We can sort a list using its sort method:

```
>>> ta = [ "Changmin", "Jeongmin", "Kyeongdeok",  
"Taesik", "Kyuho", "Sumin" ]  
>>> ta.sort()  
>>> ta  
['Changmin', 'Jeongmin', 'Kyeongdeok', 'Kyuho',  
'Sumin', 'Taesik']
```

Let's sort the medal totals: `totals.sort()`.

```
>>> totals.sort()  
>>> totals  
[(1, 'Belgium'), (1, 'Hungary'), (1, 'Ukraine'),  
(2, 'Poland'), (2, 'Slovenia'), ..., (15, 'France'),  
(17, 'Russia'), (17, 'South Korea'), ...,  
(39, 'Norway')]
```

Sequences and Structure

❑ Sequence – Lists (Slicing):

Slicing creates a new list with elements of the given list:

```
sublist = mylist[i:j]
```

Then `sublist` contains elements `i`, `i+1`, . . . `j-1` of `mylist`.

If `i` is omitted, the sublist starts with the first element.

If `j` is omitted, then the sublist ends with the last element.

Special case: We can create a copy of a list with

```
list2 = list1[:]
```

Sequences and Structure


❑ Sequence – Lists (Reversing):

We rather want the countries with the largest number of medals at the top:

```
>>> totals.reverse()  
>>> totals  
[(39, 'Norway'), (31, 'Germany'), ...,  
(17, 'South Korea'), ..., (1, 'Hungary'), (1, 'Belgium')]
```


Actually we only care about the top 10:

```
>>> top_ten = totals[:10]  
>>> for p in top_ten:  
...     medals, country = p  
...     print(medals, country)
```

 **Slicing**

We can unpack the elements in a list immediately

```
>>> for medals, country in top_ten:  
...     print(medals, country)
```

 **Unpack immediately**

Sequences and Structure

Sequence – Lists (Ranking):

Let's create the top-10 lexicographical ranking:

```
table = []
for i in range(len(countries)):
    table.append( (gold[i], silver[i], bronze[i],
                  countries[i]) )
table.sort()
top_ten = table[-10:]
top_ten.reverse()
for g, s, b, country in top_ten:
    print(country, g, s, b)
```

```
Norway 14 14 11
Germany 14 10 7
Canada 11 8 10
United States 9 8 6
Netherlands 8 6 6
Sweden 7 6 1
South Korea 5 8 4
Switzerland 5 6 4
France 5 4 6
Austria 5 3 6
```

Sequences and Structure

❑ Sequence – Lists (Selecting):

Let's find all countries that have only one kind of medal:

```
def no_medals(countries, al, bl):  
    result = []  
    for i in range(len(countries)):  
        if al[i] == 0 and bl[i] == 0:  
            result.append(countries[i])  
    return result
```

```
only_gold = no_medals(countries, silver, bronze)  
only_silver = no_medals(countries, gold, bronze)  
only_bronze = no_medals(countries, gold, silver)
```

```
only_one = only_gold + only_silver + only_bronze
```

Sequences and Structure

❑ Sequence – Lists (more ...):

List objects `L` have the following methods:

- `L.append(v)` add object `v` at the end
- `L.insert(i, v)` insert element at position `i`
- `L.pop()` remove and return last element
- `L.pop(i)` remove and return element at position `i`
- `L.remove(v)` remove first element equal to `v`
- `L.index(v)` return index of first element equal to `v`
- `L.count(v)` return number of elements equal to `v`
- `L.extend(K)` append all elements of sequence `K` to `L`
- `L.reverse()` reverse the list
- `L.sort()` sort the list

Sequences and Structure

❑ Sequence –Strings (defining):

Strings are sequences:

```
def is_palindrome(s):  
    for i in range(len(s) // 2):  
        if s[i] != s[len(s) - i - 1]:  
            return False  
    return True
```

Strings are immutable.

The `in` operator for strings:

```
>>> "abc" in "01234abcdefg"  
True  
>>> "abce" in "01234abcdefg"  
False
```

Different from the `in` operator for lists and tuples, which tests whether something is equal to an element of the list or tuple.

Sequences and Structure

❑ Sequence –Strings (method ...):

String objects have many useful methods:

- `upper()`, `lower()` and `capitalize()`
- `isalpha()` and `isdigit()`
- `startswith(prefix)` and `endswith(suffix)`
- `find(str1)`, `find(str1, start)` and `find(str1, start, end)`
- `replace(str1, str2)`
- `rstrip()`, `lstrip()` and `strip()`

String methods for converting between lists and strings:

- `split()` splits with white space as separator
- `split(sep)` splits with a given separator `sep`
- `join(list1)` concatenates strings from a list `list1`

Sequences and Structure

❑ Sequence – Lists, Tuples, Strings:

Lists and tuples are very similar, but lists are mutable, while tuples (and strings) are immutable:

We can convert a sequence into a list or tuple using the **list** and **tuple** functions:

```
>>> list(t)
['CS101', 'A+', 13]

>>> tuple(gold)
(0, 5, 2, 0, 11, 1, 2, 1, 5, ..., 7, 5, 1, 9)

>>> list("CS101")
['C', 'S', '1', '0', '1']
```

Sets and Structure

Python has a data type to implement sets in mathematics. A set is a collection of distinct objects, and therefore there can be no duplicated elements in a set.

To create a set object, we can use curly braces or the `set()` function.

```
>>> odds = {1, 3, 5, 7, 9}
>>> evens = {2, 4, 6, 8, 10}
>>> emptyset = set() # {} creates an empty dictionary
>>> randomset = {4, 6, 2, 7, 5, 2, 3} # Duplicated ele.

>>> odds
{9, 3, 5, 1, 7}
>>> evens
{8, 10, 2, 4, 6}
>>> emptyset
set()
>>> randomset
{2, 3, 4, 5, 6, 7}
```

Sets and Structure

❑ We can convert a list to a set.

```
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1,
            0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]

>>> gold
[0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1, 0, 0, 8, 11,
4, 13, 1, 2, 3, 2, 6, 1, 9]

>>> goldset = set(gold)

>>> goldset
{0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13}

>>> type(goldset)
<class 'set'>
```

We can also convert a string to set.

```
>>> set("Good morning!")
{'G', 'm', 'i', 'd', 'o', '!', 'g', 'n', 'r', ' '}
```

Sets and Structure

Python has a data type to implement sets in mathematics. A set is a collection of distinct objects, and therefore there can be no duplicated elements in a set.

A set does not have ordering, so indexing is not supported.

```
>>> odds[1]
```

```
TypeError: 'set' object does not support indexing
```

We use **in** operator for sets

```
>>> 3 in odds
```

```
True
```

```
>>> 2 in odds
```

```
False
```

```
>>> for num in odds:
```

```
...     print(num)
```

Sets and Structure

□ The set objects "s" have the following methods:

- `s.add(v)`: adds an element `v`
- `s.remove(v)`: removes an element `v`
- `s.pop()`: removes and returns an arbitrary element
- `s.intersection(k)`: returns the intersection between the sets `s` and `k` (i.e., $s \cap k$)
- `s.union(k)`: returns the union of the sets `s` and `k` (i.e., $s \cup k$)
- `s.difference(k)`: removes elements found in a set `k` (i.e., $s \cap k^c$)

Sets and Structure

▣ Examples of using the set methods:

```
>>> randomset  
{2, 3, 4, 5, 6, 7}
```

```
>>> randomset.add(9)
```

```
>>> randomset  
{2, 3, 4, 5, 6, 7, 9}
```

```
>>> randomset.remove(7)
```

```
>>> randomset  
{2, 3, 4, 5, 6, 9}
```

```
>>> randomset.pop()  
2
```

```
>>> randomset  
{3, 4, 5, 6, 9}
```

Dictionary and Structure

❑ Another useful data structure in Python is **dictionary**:

A dictionary is a collection of values. However, a dictionary can be accessed by using multiple types of indexes (i.e., not only integers, but also strings and any immutable types of objects). Indexes used for a dictionary are called keys, and a key is associated with a value. This is called a key-value pair.

To create a dictionary, we can use curly braces or the `dict()` function.

```
majors = {"CS": "Computer Science",  
          "EE": "Electrical Engineering",  
          "MAS": "Mathematical Sciences",  
          "ME": "Mechanical Engineering"}
```

```
d1 = dict() # an empty dictionary  
d2 = {} # an empty dictionary
```

Dictionary and Structure

❑ A dictionary does not have ordering, and only the keys that are defined in a dictionary can be used as an index.

```
>>> majors[0]  
KeyError: 0
```

```
majors = {"CS": "Computer Science",  
          "EE": "Electrical Engineering",  
          "MAS": "Mathematical Sciences",  
          "ME": "Mechanical Engineering"}
```

We can add a key with a value to a dictionary.

```
>>> majors["PH"] = "Physic"  
>>> majors["PH"]  
'Physic'
```

We can also change the value via the key in the dictionary.

```
>>> majors["PH"] = "Physics"  
>>> majors["PH"]  
'Physics'
```

Dictionary and Structure

□ A dictionary object "d" has the following methods and operators:

- `len(d)`: returns the number of elements in d
- `key in d`: returns **True** if d has the key, otherwise returns **False**
- `d.get(key, default=None)`: Returns the value that corresponds to the key, or returns the `default` value if the key is not defined in d)
- `d.keys()`: returns a list of keys in d
- `d.values()`: returns a list of values in d
- `d.items()`: returns a list of key-value pairs in d
- **del** `d[key]`: removes the key-value pair that corresponds to the key

The objects that are returned from `keys()`, `values()` and `items()` are not list objects.

They have elements like lists, but they cannot be modified and do not have an `append()` method.

Dictionary and Structure

▣ Examples of using the dictionary methods:

```
>>> majors
{0: 0.001, 'CS': 'Computer Science', 'PH': 'Physics',
'ME': 'Mechanical Engineering', 'EE': 'Electrical Engineering',
'MAS': 'Mathematical Sciences'}
>>> len(majors)

>>> del majors[0]
>>> majors
{'CS': 'Computer Science', 'PH': 'Physics',
'ME': 'Mechanical Engineering', 'EE': 'Electrical Engineering',
'MAS': 'Mathematical Sciences'}
>>> len(majors)

>>> "CS" in majors
True
>>> "ELS" in majors
False
```

Dictionary and Structure

▣ Examples of using the dictionary methods:

```
>>> majors
{0: 0.001, 'CS': 'Computer Science', 'PH': 'Physics',
 'ME': 'Mechanical Engineering', 'EE': 'Electrical Engineering',
 'MAS': 'Mathematical Sciences'}
>>> len(majors)
```

```
>>> majors.keys()
dict_keys(['CS', 'PH', 'ME', 'EE', 'MAS'])
```

```
>>> majors.values()
dict_values(['Computer Science', 'Physics', 'Mechanical
Engineering', 'Electrical Engineering', 'Mathematical
Sciences'])
```

```
>>> majors.items()
dict_items([('CS', 'Computer Science'), ('PH', 'Physics'),
 ('ME', 'Mechanical Engineering'), ('EE', 'Electrical Engineering'),
 ('MAS', 'Mathematical Sciences')])
```

Dictionary and Structure

❑ Loop in a dictionary:

To loop over the keys in adictionary, we can use the in operator

```
>>> for key in majors:
...     print("%s is %s." % (key, majors[key]))
CS is Computer Science.
PH is Physics.
ME is Mechanical Engineering.
EE is Electrical Engineering.
MAS is Mathematical Sciences.
```

To loop over both keys and values in a dictionary, we can use items()

```
>>> for key, value in majors.items():
...     print("%s is %s." % (key, value))
CS is Computer Science.
PH is Physics.
ME is Mechanical Engineering.
EE is Electrical Engineering.
MAS is Mathematical Sciences.
```


List, Set and Dictionary

❑ When do we use list, set or dictionary?

- If we need to manage an ordered sequence of objects
→ Use a **List**
- If we need to manage an unordered set of values
→ Use a **Set**
- If we need to associate values with keys, so that we can easily look up the values by the keys
→ Use a **Dictionary**

List, Set and Dictionary

- ❑ Using a set is more efficient than using a list when we check membership of a value.

```
import time
large_list = list(range(10000000))
large_set = set(large_list)

st = time.time()
for num in range(100000):
    if num not in large_list:
        print("What?!")
print("Running time for list: %f sec" % (time.time() - st))

st = time.time()
for num in range(100000):
    if num not in large_set:
        print("What?!")
print("Running time for set: %f sec" % (time.time() - st))
```

Arrays and Structure

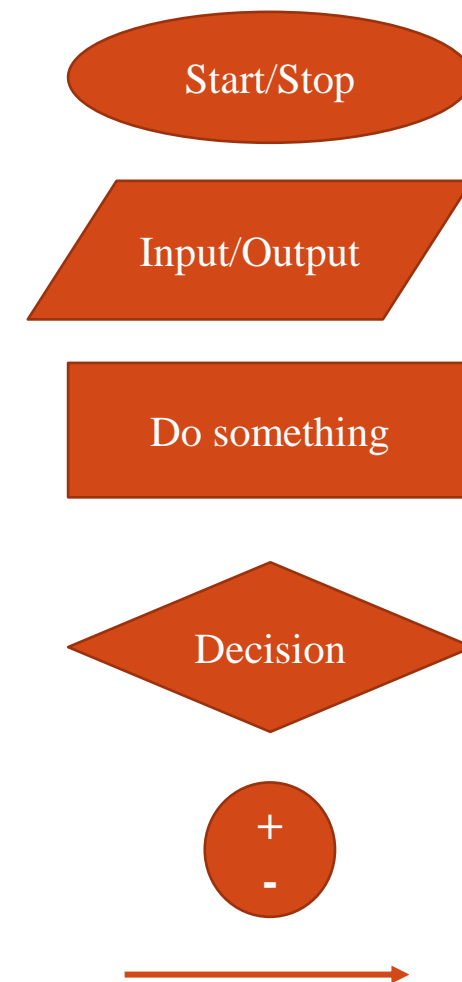
Next week ...

Bài tập thực hành



Viết một chương trình thống kê và tính toán, xuất nhập dữ liệu bằng dữ liệu các quốc gia. (*theo bảng số liệu kết quả thể thao trong slide*)

- Tính tổng số huy chương vàng của các quốc gia
(2 Silver = 1 Gold, 3 Bronze = 1 Gold)
- Sắp xếp 1st, 2nd, 3rd thứ hạng quốc gia (theo số huy chương vàng)
- Tạo list thứ hạng, xuất thành file lưu vào máy tính.



Bài tập thực hành

❑ Xây dựng flow chart (algorithm):

- Xây dựng bảng dữ liệu
→ file excel → file .csv
- Load file dữ liệu, truy xuất dữ liệu cho chương trình
- Thống kê, Tính toán, So sánh:
 - Tính số Gold mỗi quốc gia theo quy luật đã cho
 - Tạo lệnh so sánh số Gold các quốc gia để sắp xếp thứ hạng
- Tạo list danh sách thứ hạng 1st, 2nd, 3rd
- Xuất dữ liệu đã tính toán, sắp xếp
- Tạo file dữ liệu, save as vào máy tính
- Kết thúc

