# Python Basics

NGUYEN Hong Thinh

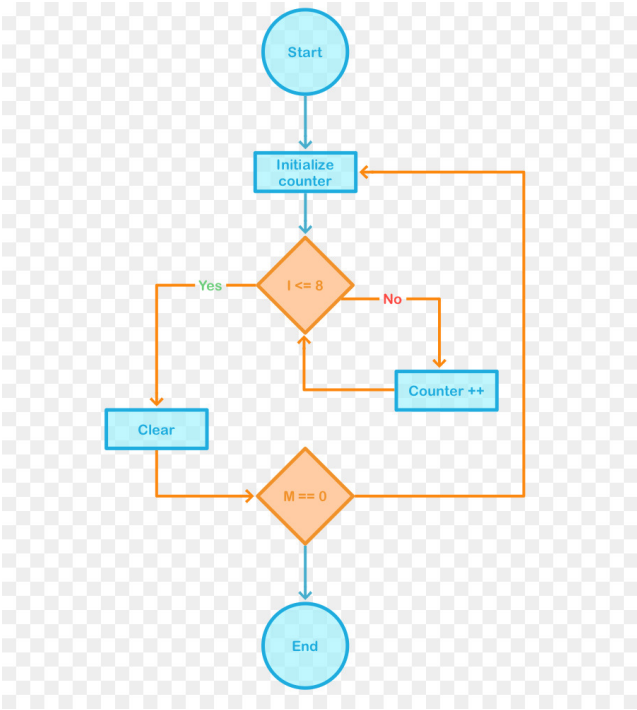
FET-UET-VNU

Ngày 25 tháng 11 năm 2019

# Program là gì?



- Program là chuỗi các chỉ dẫn để thực thi một tác vụ nào đấy.
- Program ở các ngôn ngữ lập trình khác nhau, sẽ có cách viết khác nhau, nhưng thường gồm 4 phần chính
  - Variables (Biến số)
  - Math: các phép toán giữa các biến
  - Conditional execution: điều kiện thực thi
  - Repetition: Lặp lại hành động



# Điều kiện thực thi:



- Sử dụng cấu trúc If/Elif/Else:

```
1 if(a>b):
2   print("greater")
3 elif(a<b):
4   print("lower")
5 else:
6   print("equal")
7
8
```

- Có thể có nhiều elif ở giữa If và Else
- Dùng tab để phân biệt các đoạn lệnh điều kiện

# Repetition –Lặp



- Có 2 cách thực hiện lệnh lặp:
- Dùng for...in

```
1 for i in range(0,10):
2   print(i)
3
```

- Dùng while:

```
1 j=0
2 while (j<10):
3   print(j)
```

# Biến số và phép toán



- Từng kiểu biến sẽ định nghĩa các phép toán và hàm thực thi khác nhau
- Các kiểu biến phổ biến: (immutable = un-modifiable: không thể thay đổi)

| Class | Description | Immutable? |
|---|---|---|
| **bool** | Boolean value | ✓ |
| **int** | integer (arbitrary magnitude) | ✓ |
| **float** | floating-point number | ✓ |
| **list** | mutable sequence of objects | |
| **tuple** | immutable sequence of objects | ✓ |
| **str** | character string | ✓ |
| **set** | unordered set of distinct objects | |
| **frozenset** | immutable form of set class | ✓ |
| **dict** | associative mapping (aka dictionary) | |

# Khởi tạo biến



- Python không yêu cầu định nghĩa kiểu biến trước khai báo. Tuỳ vào giá trị được khởi tạo kiểu gì, biến có kiểu đó
- Dùng lệnh type để kiểm tra kiểu biến

```
1 a=1.2 ; type(a)
2 b='hello'; type(b)
3 a=b;type(a)
```

- Có thể ép kiểu:

```
1 a=1.22; b= int(a)
```

- Tên biến:
  - Có thể chứa ký tự alphabet, _, hoặc số, tuy nhiên không thể bắt đầu bằng số.
  - Không dùng keyword làm tên biến.
  - Phân biệt chữ hoa chữ thường

# Biến số học (int/float) và phép toán số học



Hãy thử lần lượt các phép toán sau trên Jupyter và rút ra kết luận:

```
3+6
6/2
13-5
4*2
4**3
16//5
16%5
5--4
4++3
+3-2
```

# Phép toán số học

## Arithmetic Operators

| Operator | Meaning | Example |
|---|---|---|
| + | Addition | 4 + 7 ⟶ 11 |
| - | Subtraction | 12 - 5 ⟶ 7 |
| * | Multiplication | 6 * 6 ⟶ 36 |
| / | Division | 30 / 5 ⟶ 6 |
| % | Modulus | 10 % 4 ⟶ 2 |
| // | Quotient | 18 // 5 ⟶ 3 |
| ** | Exponent | 3 ** 5 ⟶ 243 |

# Quy tắc tính toán



- Các biểu thức tính toán trên Python tuân theo quy tắc toán học thông thường:
- Dấu ngoặc đơn ưu tiên cao nhất
- Phép hàm mũ có mức ưu tiên thứ 2
- Phép nhân chia có mức ưu tiên như nhau và ở mức 3
- Cộng trừ có mức ưu tiên thấp nhất
- Quy tắc từ trái sang phải với các phép có ưu tiên như nhau
- An toàn: hãy sử dụng dấu ngoặc đơn

```
1  3+3**2+2*3-2/3+3%3
2  3**2/6+3-3/4+4
3
```

# Biến Boolean (bool) và phép toán logic



- Các biến Boolean hay biến logic chỉ nhận 2 giá trị True và False
- Thường là kết quả của phép toán Boolean (phép toán quan hệ) và được sử dụng trong các lệnh điều kiện:

| Operator | Description |
|---|---|
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| == | Equal to |
| != | Not equal to |

# Biến nhị phân (binary) và phép toán trên từng bit



- Các biến nhị phân được biểu diễn chỉ bởi 0-1
- Các phép toán cơ bản trên biến nhị phân:

| Operator | Description |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise XOR |
| ~ | Bitwise NOT |
| >> | Bitwise right shift |
| << | Bitwise left shift |

# Phép gán



- Phép gán (=) được sử dụng để khởi tạo và thay đổi giá trị của biến
- Một số phép gán thay đổi giá trị của cùng biến số, có thể viết tắt:

| Operator | Example | Equals To |
|---|---|---|
| = | a = 10 | a = 10 |
| += | a += 10 | a = a+10 |
| -= | a -= 10 | a = a-10 |
| *= | a *= 10 | a = a*10 |
| /= | a /= 10 | a = a / 10 |
| %= | a %= 10 | a = a % 10 |
| //= | a //= 10 | a = a // 10 |
| **= | a **= 10 | a = a ** 10 |
| &= | a &= 10 | a = a & 10 |
| \|= | a \|= 10 | a = a \|10 |
| ^= | a ^= 10 | a = a ^10 |
| >>= | a >>= 10 | a = a >> 10 |
| <<= | a <<= 10 | a = a << 10 |

# Phép gán



- Có thể gán nhiều biến đồng thời trên cùng dòng lệnh:

```
1     x, y = 2, 3
2
```

- Lệnh swap giá trị cũng có thể thực hiện nhanh:

```
1     x, y = y,x
```

- Có thể thực hiện lệnh gán liên tiếp:

```
1     x, y,z = 1,2,3
```

# Bài tập số học:

1. Viết chương trình tính ước chung lớn nhất và bội chung nhỏ nhất của 2 số nguyên dương cho trước
2. Viết chương trình in ra dãy số Fibonaci bé hơn một số nguyên N cho trước. Biết dãy Fibonaci: 1,1,2,3,5,8,13,21....
3. Viết chương trình in ra tất cả các số nguyên tố bé hơn một số nguyên N cho trước.
4. Viết hàm tính giai thừa (sử dụng đệ quy và lặp)

# Bài tập FOR-IFELSE:



Cước taxi: Viết chương trình tính cước taxi, khi biết quãng đường đã đi S. Biết giá cước cho từng đoạn giá

1. Nếu S<0.5 km: Cước =10,000 /km
2. với 0.5<=S<5 km: Cước =9,000 /km
3. với 5<= S< 10 km: Cước =8,000 /km
4. Nếu 10<= S< 20 km: Cước =7,000 /km
5. Nếu 20<= S< 30 km: Cước =6,000 /km
6. Nếu S >= 30 km: Cước =5,000 /km

# String: Kiểu chuỗi



- Sử dụng 2 dấu ’ hoặc 2 dấu " để khai báo 1 string

```
1       a='hello'
2       b="hello"
3
```

- Bản thân “ và ‘ khác nhau về mặt giá trị, khi ở trong string:

```
1     a="Say, 'Hello, Jeremy' "
2     b='Say, "Hello, Jeremy" '
3     a==b #False
```

- Muốn hiển thị ký tự đặc biệt trong string, sử dụng \ trước kí tự đặc biệt đó
- Một vài ký tự có ý nghĩa đặc biệt: \n (new line), \t (new tab), \r (raw string)

# Các phép toán trên string:



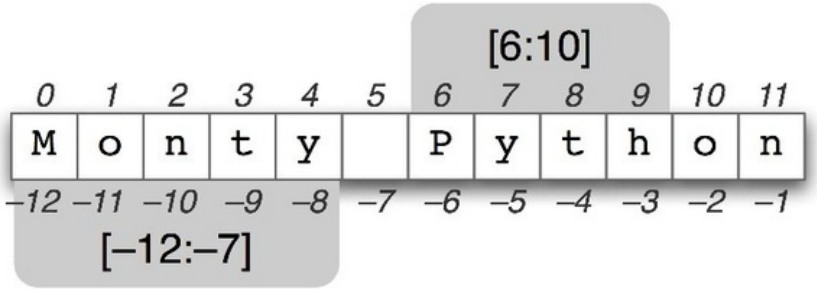
- Phép + (concat): ghép 2 string với nhau tạo ra 1 string mới dài hơn
- Phép * (repeat): lặp lại string đó n lần, tạo ra 1 string mới dài hơn
- in/not in: để kiểm tra 1 string có trong string khác không
- is/is not: để kiểm trasự tương đồng giữa các string:

```
1     s0="Hello"
2     s1=s0+"How are you";s1
3     s2=s0*3;s2
4     s0 in s1
5     s0 not in s1
6     s1 is s2
```

# Index và Slicing



- Có thể coi String là 1 vector mà mỗi thành phần của nó là các ký tự char. Khi đó mỗi ký tự sẽ có vị trí được gán chỉ số vị trí (index).




- Có thể lấy ký tự tại 1 vị trí cụ thể, sử dụng [ ].

```
1    s="Hello"
2    s[0]
3
```

- Để lấy giá trị của string tại 1 nhóm các chỉ số liên tiếp từ start-index đến end-index , sử dụng dấu **start-index : end-index+1**. Phương pháp này gọi là slicing

```
1
2    s[0:3]
3
```

# Immutable - Bất biến, không thể sửa đổi



- Với kiểu string, ta không thể thay đổi từng thành phần của nó được.

```
1       a="hello"
2       a[0]='H' #error
3
```

- Các phép toán trên nó đều sẽ sinh ra 1 string mới.
- Một vài hàm đặc biệt với string: **.lower(), .upper, .split(), .replace()**

```
1       s="AbscQAf"
2       s1=s.lower()
3       s2=s.upper()
4       s3=s.replace('A','aa')
5       s.split()
6
```

# Một vài hàm đặc biệt với string:



- Chuyển ký tự thành số và số thành ký tự: **ord(), chr()**

```
1       ord("a") # 97
2       chr(98) # 'b'
3
```

- Kiểm tra ký tự viết hoa/viết thường: isupper(), islower():

```
1
2         'A'.isupper()
3         'a'.islower()
4
```

# Bài tập String:



1. Viết chương trình kiểm tra xem trong chuỗi nhập vào có ký tự nào in hoa không. Nếu có trả về True, và vị trí đầu tiên xuất hiện chữ hoa
2. 1 từ tiếng Anh được gọi là "abecedarian" nếu các ký tự của nó đảm bảo theo thứ tự alphabet. VD ader, benq Hãy viết chương trình kiểm tra xem một từ nhập vào bàn phím có là "abecedarian" không?
3. Mã hoá vòng: Một bộ mã hoá thực hiện chuyển đổi vòng các ký tự alphabet theo 1 số N bước cho trước. Giả sử mã hoá N=+2: a→c, z→b. . . Với mã N=-1 thì a→z, b→a. . . Viết chương trình chuyển đổi 1 string thành mã vòng của nó, với N biết trước

# Kiểu Set -tập hợp



- Set là 1 tập các item, không có thứ tự, khác nhau. Nói cách khác, mọi item trong set đều là **duy nhất**
- Các item của set có thể nhận mọi kiểu giá trị. Có 2 loại set: frozenset (cố định, không thể thay đổi) và MutableSet (hoặc Set) (có thể thêm bớt phần tử).
- Set thường được sử dụng với string, với lệnh lặp

```
1  from sets import Set
2  Kythuat = Set(['Tam', 'Ngoc', 'Nam', 'Lan'])
3  Laptrinh = Set(['Nam', 'Sam', 'An', 'Lan'])
4  Quanly = Set(['Ngoc', 'Nam', 'An', 'Trung'])
5  Nhanvien = Kythuat | Laptrinh | Quanly
         # Phep hop
6  Quanly_kythuat = Kythuat & Quanly
     # phep giao
7  Quanlychung = Quanly - Kythuat - Laptrinh #
   difference
```

# Một số phép toán trên Set



| Operation | Equivalent | Result |
|---|---|---|
| `len(s)` | | number of elements in set *s* (cardinality) |
| `x in s` | | test *x* for membership in *s* |
| `x not in s` | | test *x* for non-membership in *s* |
| `s.issubset(t)` | `s <= t` | test whether every element in *s* is in *t* |
| `s.issuperset(t)` | `s >= t` | test whether every element in *t* is in *s* |
| `s.union(t)` | `s \| t` | new set with elements from both *s* and *t* |
| `s.intersection(t)` | `s & t` | new set with elements common to *s* and *t* |
| `s.difference(t)` | `s - t` | new set with elements in *s* but not in *t* |
| `s.symmetric_difference(t)` | `s ^ t` | new set with elements in either *s* or *t* but not both |
| `s.copy()` | | new set with a shallow copy of *s* |

# Một số phép toán trên Set



| Operation | Equivalent | Result |
|---|---|---|
| `s.update(t)` | *s* \|= *t* | return set *s* with elements added from *t* |
| `s.intersection_update(t)` | *s* &= *t* | return set *s* keeping only elements also found in *t* |
| `s.difference_update(t)` | *s* -= *t* | return set *s* after removing elements found in *t* |
| `s.symmetric_difference_update(t)` | *s* ^= *t* | return set *s* with elements from *s* or *t* but not both |
| `s.add(x)` | | add element *x* to set *s* |
| `s.remove(x)` | | remove *x* from set *s*; raises `KeyError` if not present |
| `s.discard(x)` | | removes *x* from set *s* if present |
| `s.pop()` | | remove and return an arbitrary element from *s*; raises `KeyError` if empty |
| `s.clear()` | | remove all elements from set *s* |

# Kiểu Tuple và List



- Tuple và List: đều là một chuỗi (sequence) các item. Khác với string là chuỗi của chỉ các ký tự (char), các item của Tuple hay List có thể là bất cứ kiểu gì, và không cần phải duy nhất (như Set)
- Khai báo:
  - Tuple: sử dụng ( ) và dấu , để ngăn cách các item
  - List: sử dụng [ ] và dấu , để ngăn cách các item

```
1 a=['hello', 10, 9.01, [1,1],(1,3)];
2 type(a)
3 b=('hello', 10, 9.01, [1,2],(1,3))
4 type(b)
```

# Kiểu Tuple và List



- Khác biệt lớn nhất của Tuple và List là khả năng sửa đổi (tính bất biến).
- Tuple giống với string, immutable, không thể thay đổi, thêm bớt giá trị.
- Lists có thể thay đổi (mutable). Nói cách khác: Tuple có thể coi là frozen-list (1 list cố định)

```
1 a=['hello', 10, 9.01, [1,1],(1,3)];
2 a[0]=5# ok
3 b=('hello', 10, 9.01, [1,2],(1,3))
4 b[0]=5#Error
```

# Index and Slicing



- Các item của Tuple và List đều có thứ tự và được gán index.
- Giống với string, ta có thể sử dụng [] tại index để lấy giá trị và slicing để nhóm các item liên tiếp:

```
1 a=['hello', 10, 9.01, [1,1],(1,3)];
2 a[2]
3 a[0:5]
4 b=('hello', 10, 9.01, [1,2],(1,3))
5 b[-2:-4]
```

# Một số phép toán trên List



```
.append(x)
.insert(i, x)
.remove(x)
.pop([i ])
.clear()
.index()
.count(x)
.sort()
.reverse()
.copy()
```

# Một số phép toán trên List



```
1  a=[] # initial
2  a. append(1.1); a
3  a. append("Hello"); a
4
5  fruits = ['orange', 'apple', 'pear','banana', '
       kiwi', 'apple']
6  fruits.count('apple')
7  fruits.count('pineapple')
8  fruits.index('banana')
9  fruits.reverse()
10 fruits.append('grape')
11 fruits.sort()
12 fruits.pop()
```

# các hàm số trên tuple và list



- Phép +: để ghép các list hoặc tuple
- Phép *: lặp lại n lần
- In/not in/is/is not: Giống với string

```
1 v=(1, 2, 3) * 3
2 u=[1, 2, 3] +[2,3,4,5,6]
```

# Bài tập List



Bag of word for document:

1. Đưa vào 1 đoạn văn bản dài S
2. Hãy xác định số từ của S?
3. Hãy loại bỏ các ký tự đặc biệt, chuyển sang chữ thường.
4. Hãy xác định các từ khác nhau của S? Có bao nhiêu từ khác nhau?
5. Đếm số lượng các từ đó ta gọi đó là (histogram của văn bản)

# Bài tập List

Bag of word for document

1. Đưa vào một vài đoạn văn bản dài S
2. Hãy xác định tập tất cả các từ của các văn bản.
3. Gọi tập này là dictionary. Hãy đếm xem trên mỗi văn bản, các từ xuất hiện bao nhiêu lần.
4. Với các từ xuất hiện ít hơn 3 lần, hãy loại bỏ nó.
5. Biểu diễn lại các văn bản thành histogram của các từ còn lại

# Bài tập List

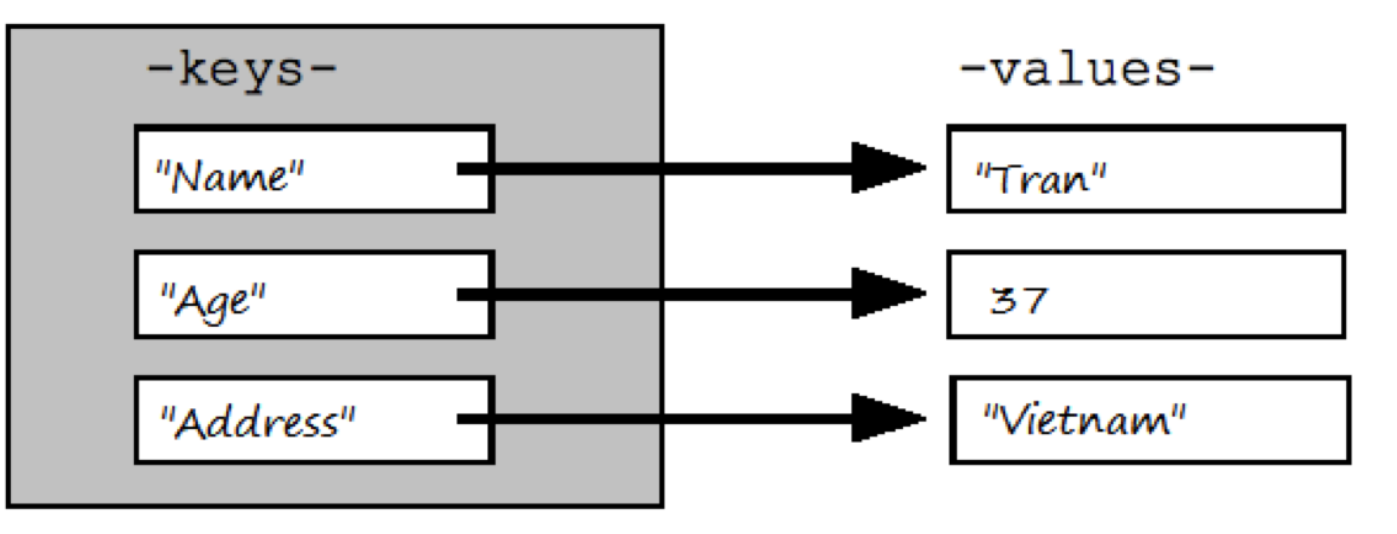


1. Sử dụng kiểu list để lưu dữ liệu kiểu [[SV, điểm], [SV, điểm], [SV, điểm]..]
2. Nhập điểm thi SV từ bàn phím
3. Viết chương trình hiển thị tên và điểm SV theo thứ tự tên SV
4. Viết chương trình hiển thị tên và điểm SV theo thứ tự điểm từ cao xuống thấp
5. Viết chương trình hiển thị tất cả các SV bị trượt (điểm thi $<5$)

# Kiểu Dictionary - Từ điển



- Có thể coi là list mở rộng:
  - List: các thành phần của list có thể thuộc bất cứ kiểu gì, với chỉ số (index) là các số nguyên
  - Dict: Chỉ số (index) không cần phải là số nguyên; mà là các key (key có thể thuộc bất cứ kiểu gì)
- Dict thực hiện việc map các key với các values



# Đặc điểm của kiểu Dictionary



- 1 Dictionary gồm 2 thành phần: **Keys** và **Values**
- **Key** là duy nhất
- Các Key có thể nhận nhiều kiểu dữ liệu khác nhau, nhưng không thể thay đổi Q: Key có thể nhận dữ liệu thuộc các kiểu nào ?? (number, string, tuple, list)
- **Values**: được map tương ứng với các Key, có thể nhận bất cứ kiểu dữ liệu nào

# Khai báo kiểu Dictionary



```
1  Prices={'orrange': 50, 'apple':65, 'banana' : 32}
2  # => {'orrange': 50, 'apple': 65, 'banana': 32}
3
4  infor = dict(name = 'Tran', age = 30,add= 'Hanoi')
5  #=> {'name': 'Nguyen', 'age': 33, 'add': 'Hanoi'}
6
7  weather=dict()
8  weather['morning']='sunny'
9  weather['affternoon']=['cloudy', 'rainy']
10 #=> {'morning': 'sunny', 'affternoon': ['cloudy',
       'rainy']}
```

- Có 3 cách khác nhau để khai báo 1 dict

# Trích xuất ra các phần tử của Dictionary:



- List, Tuple hay String dùng index để lấy giá trị các phần tử s[index]
- Dictionary sử dụng Key để trích xuất giá trị

```
Prices={'orrange': 50, 'apple':65}
Prices['orrange']
# 50
```

- Ngoài ra cũng có thể dùng hàm **.get()** theo các key để lấy giá trị:

```
Prices.get('orrange')
# 50
```

# Cập nhật Dictionary:



```
1 Prices={'orrange': 50, 'apple':65}
2 Prices['orrange'] =100
3 Prices
4 # {'orrange': 100, 'apple':65}
5
6 Prices['grape'] =41;
7 Prices
8 #{grape':41, 'orrange': 100, 'apple':65}
```

- Sử dụng phép gán, sẽ thêm 1 cặp Key-Values vào Dict hoặc thay đổi giá trị của một Key nếu Key đó đã tồn tại
- Cộng thêm giá trị vào values?

```
1 Prices['grape'] +=20
```

# Xoá 1 Key trong Dictionary:



- Có nhiều cách để xoá một Key:

```
Prices={'orrange': 50, 'apple':65, 'grape':
    61}
del Prices['orrange']

Prices.__delitem__('apple')


Prices.pop('grape')
```

- Cũng có thể xoá cả từ điển, sử dụng hàm **del**

```
del Prices
```

# Một vài hàm thông dụng với Dictionary:



| Method | Description |
|---|---|
| clear() | Removes all the elements from the dictionary |
| copy() | Returns a copy of the dictionary |
| fromkeys() | Returns a dictionary with the specified keys and values |
| get() | Returns the value of the specified key |
| items() | Returns a list containing the a tuple for each key value pair |
| keys() | Returns a list containing the dictionary's keys |
| pop() | Removes the element with the specified key |
| popitem() | Removes the last inserted key-value pair |
| setdefault() | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| update() | Updates the dictionary with the specified key-value pairs |

# Một vài hàm thông dụng với Dictionary



- Xác định chiều dài: **len()**

```
len(Prices)
```

- Xác định tất cả các values và các keys: **.values(), keys()**

```
Prices.values()
Prices.keys()

for keyname in Prices:
  print(keyname)
```

- Xác định 1 key có trong Dictionary không? in

```
'orrange' in Prices
'Mango' in Prices
```

- Xác định cả key và values Dictionary: **.items()**

```
Prices.items()
```

# Một vài hàm thông dụng với Dictionary



- Xác định chiều dài: **len()**

```
len(Prices)
```

- Xác định tất cả các values và các keys: **.values(), keys()**

```
Prices.values()
Prices.keys()

for keyname in Prices:
  print(keyname)
```

- Xác định 1 key có trong Dictionary không? in

```
'orrange' in Prices
'Mango' in Prices
```

- Xác định cả key và values Dictionary: **.items()**

```
Prices.items()
```

# Bài tập Dictionary



Cho bảng dữ liệu:

| Fruit | Price (/Kg) | Stock(Kg) |
|---|---|---|
| Tao | 120 | 14 |
| Cam | 56 | 12 |
| Buoi | 68 | 13 |
| Chuoi | 25 | 15 |
| Xoai | 50 | 10 |

1. Lưu bảng trên theo kiểu Dict (dùng 1 hoặc 2 Dict đều được)
2. Viết hàm nhập thêm phần vào kho: VD Nho,80, 20
3. Viết chương trình hiển thị thông tin của kho: Loại quả, số lượng

# Bài tập Dictionary



1. Bây giờ, giả sử có các order mua hàng. Mỗi order sẽ hợp lệ nếu tất cả các yêu cầu (loại quả) người dùng mua, cửa hàng đều có và số lượng đáp ứng được.
2. Kiểm tra order và thông báo nếu order không hợp lệ
3. Tính số tiền người mua phải trả cho order hợp lệ
4. Cập nhật lại kho sau mỗi lần bán thành công
   order1: 12/08/2019 12:00:00 Cam 10, Tao 5, Chuoi 15
   order2: 12/08/2019 12:30:01 Tao 6, Xoai 9