https://www.facebook.com/hcmut.ml.iot.lab

Lesson 1

Introdution to Python

ML IoT

HCMUT EE MACHINE LEARNING & IOT LAB

MSc. Nguyen Khanh Loi
Ho Chi Minh City University of Technology

4/2021

DATA STRUCTURES

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Data types

- **Numeric Types**: int, float and complex
- **Boolean**: True, False
- **String**: is sequence of unicode characters.
- **List/Tuple**: is an ordered sequence of items, only difference is mutable and immutable.
- **Set**: is an unordered collection of unique items.
- **Dict**: is an unordered collection of unique items.
- **None**: is used to define a null value, or no value at all

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Number

➢ Integer

```
>>> print(123123123123123123123123123123123123123123123123 + 1)
123123123123123123123123123123123123123123123123124
```

➢ Float

```
>>> 4.2
4.2
>>> type(4.2)
<class 'float'>
>>> 4.
4.0
>>> .2
0.2

>>> .4e7
4000000.0
>>> type(.4e7)
<class 'float'>
>>> 4.2e-4
0.00042
```

➢ Complex Numbers

```
>>> 2+3j
(2+3j)
>>> type(2+3j)
<class 'complex'>
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Number - Operation

Operation	Result
`x + y`	sum of x and y
`x - y`	difference of x and y
`x * y`	product of x and y
`x / y`	quotient of x and y
`x // y`	floored quotient of x and y
`x % y`	remainder of x / y
`-x`	x negated
`+x`	x unchanged
`abs(x)`	absolute value or magnitude of x
`int(x)`	x converted to integer
`float(x)`	x converted to floating point
`complex(re, im)`	a complex number with real part re, imaginary part im. im defaults to zero.
`c.conjugate()`	conjugate of the complex number c
`divmod(x, y)`	the pair (x // y, x % y)
`pow(x, y)`	x to the power y
`x ** y`	x to the power y

https://www.facebook.com/hcmut.ml.iot.lab
HCMUT EE Machine Learning & IoT Lab

Number - Operation

```
import math
```

Operation	Result
`math.trunc(x)`	x truncated to Integral
`round(x[, n])`	x rounded to n digits, rounding half to even. If n is omitted, it defaults to 0.
`math.floor(x)`	the greatest Integral <= x
`math.ceil(x)`	the least Integral >= x

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Boolean

Vicohub

➢ Boolean

Booleans represent one of two values: True or False.

```
>>> print(True)
True

>>> print(False)
False

>>> print(type(True))
<class 'bool'>
```

Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then True, else False

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity
is not	negated object identity

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Sequence Types

- **List:** are mutable sequences. Ordered sequence of items of mixed types.

```
>>> a = ["apples", 1, "oranges"]
>>> a[0]=2
>>> a
[2, 1, 'oranges']
```

- **Tuple:** are immutable sequences. Similarly as List but cannot modify.

```
>>> a = ("apples", 1, "oranges")
>>> a[0]=2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> a
('apples', 1, 'oranges')
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Sequence Types

- **Range**: The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops..

```
class range(start, stop[, step])
```

```
>>> a = range(1,10,2)
>>> a range(1, 10, 2)
>>> list(a)
[1, 3, 5, 7, 9]

>>> a[1]
3

>>> a[1] = 2
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: 'range' object does not support
item assignment
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Text Sequence Type: String

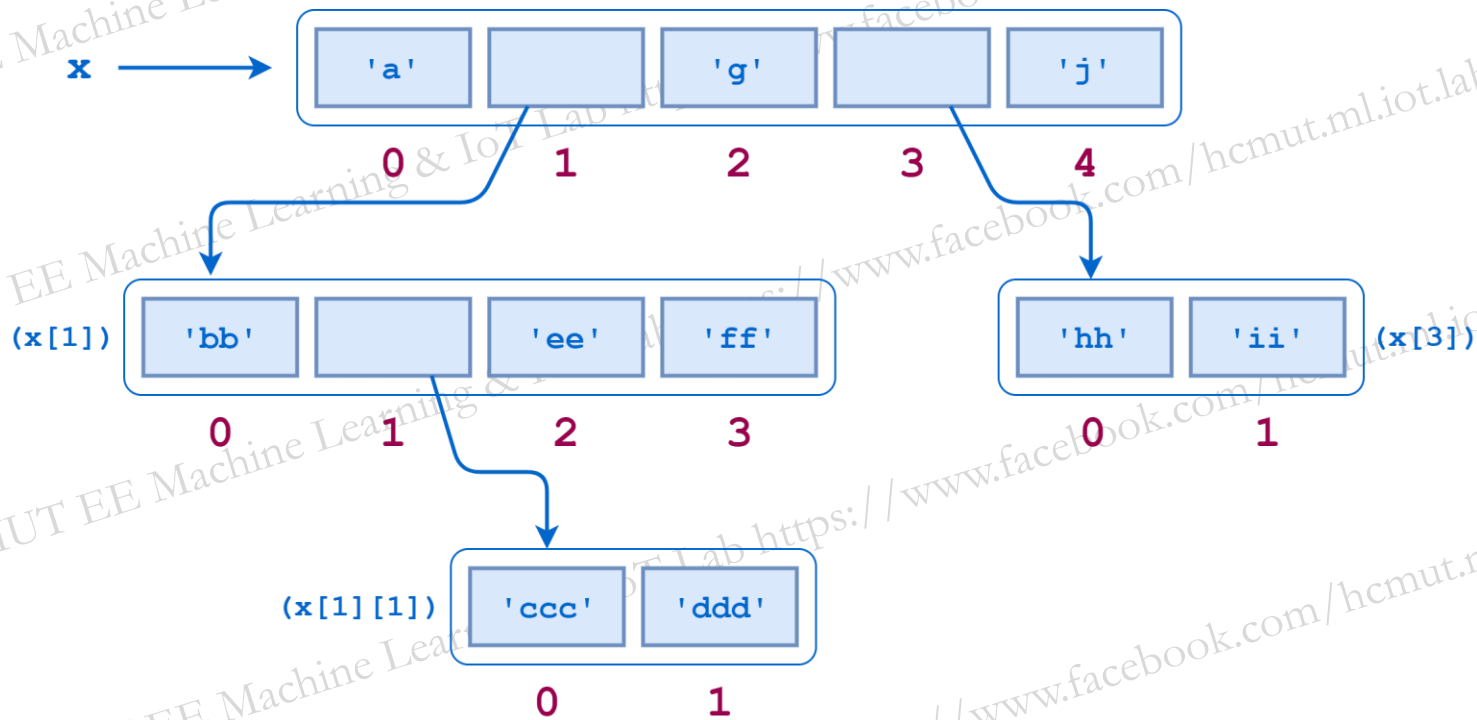
Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:

- Single quotes: `'allows embedded "double" quotes'`
- Double quotes: `"allows embedded 'single' quotes"`.
- Triple quoted:
  - `'''Three single quotes'''`
  - `"""Three double quotes"""`

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

List nested

```
>>> x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
>>> x
['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
```



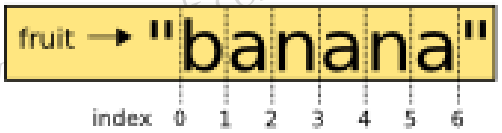
HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Slices

```
>>> singers = "Peter, Paul, and Mary"
>>> singers[0:5]
'Peter'
>>> singers[7:11]
'Paul'
>>> singers[17:21]
'Mary'
>>> classmates = ("Alejandro", "Ed", "Kathryn", "Presila", "Sean",
"Peter")
>>> classmates[2:4]
('Kathryn', 'Presila')
```

The operator [n:m] returns the part of the sequence from the n'th element to the m'th element, including the first but excluding the last.

```
>>> fruit = "banana"
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```



What do you think s[:] or s[-2:] means?

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Slices

```
>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
>>> a[0]
'foo'
>>> a[2]
'baz'
>>> a[5]
'corge'
```

'foo'	'bar'	'baz'	'qux'	'quux'	'corge'
0	1	2	3	4	5

```
>>> a[-1]
'corge'
>>> a[-2]
'quux'
>>> a[-5]
'bar'
>>> a[2:5]
['baz', 'qux', 'quux']
>>> a[-5:-2]
['bar', 'baz', 'qux']
>>> a[1:4]
['bar', 'baz', 'qux']
>>> a[-5:-2] == a[1:4]
True
```

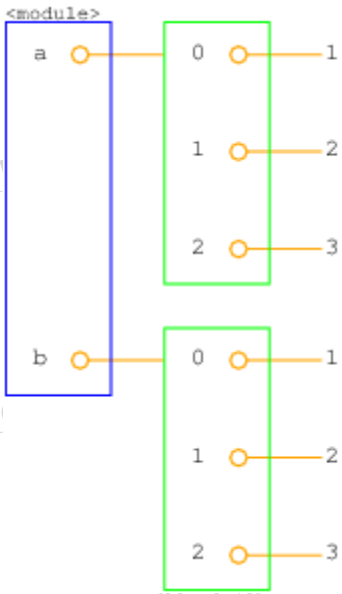
-6	-5	-4	-3	-2	-1
'foo'	'bar'	'baz'	'qux'	'quux'	'corge'
0	1	2	3	4	5

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Names, Aliasing & Cloning lists

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a == b
True

>>> a is b
False
```

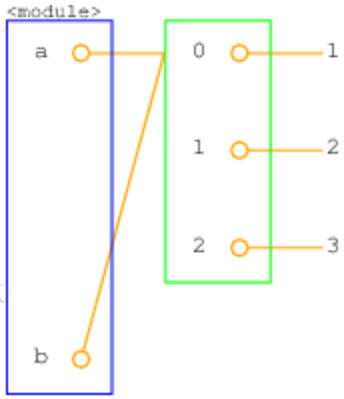



```
>>> a = [1, 2, 3]
>>> b = a
>>> a is b
True

>>> b[0] = 5
>>> a
[5, 2, 3]

>>> a = [1, 2, 3]
>>> b = a[:]
>>> b
[1, 2, 3]

>>> b[0] = 5
>>> a
[1, 2, 3]
```



HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Common Sequence Operations

Operation	Result
`x in s`	True if an item of *s* is equal to *x*, else False
`x not in s`	False if an item of *s* is equal to *x*, else True
`s + t`	the concatenation of *s* and *t*
`s * n or n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Common Sequence Operations

```
>>> 1 in (1,2,3) True
>>> lists = [[]] * 3
>>> lists
[[], [], []]

>>> lists[0].append(3)
>>> lists
[[3], [3], [3]]

>>> lists = lists * 3
>>> lists
[[3], [3], [3], [3], [3], [3], [3], [3], [3]]

>>> tuples = (1,2,3,4,5,6)
>>> tuples[2]
3

>>> tuples[0]
1

>>> tuples.count(1) 1
>>> tuples.index(3) 2
>>> max(tuples) 6
>>> min(tuples) 1
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Mutable Sequence Operations

Operation	Result
`s[i] = x`	item *i* of *s* is replaced by *x*
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
`del s[i:j]`	same as s[i:j] = []
`s[i:j:k] = t`	the elements of s[i:j:k] are replaced by those of *t*
`del s[i:j:k]`	removes the elements of s[i:j:k] from the list
`s.append(x)`	appends *x* to the end of the sequence (same as s[len(s):len(s)] = [x])
`s.clear()`	removes all items from *s* (same as del s[:])
`s.copy()`	creates a shallow copy of *s* (same as s[:])
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as s[len(s):len(s)] = t)
`s *= n`	updates *s* with its contents repeated *n* times
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as s[i:i] = [x])
`s.pop([i])`	retrieves the item at *i* and also removes it from *s*
`s.remove(x)`	remove the first item from *s* where s[i] is equal to *x*
`s.reverse()`	reverses the items of *s* in place

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Mutable Sequence Operations

```
>>> s = [1,2,"three", (4,5), [6,7], 8, 9]
>>> s[6] = "nine"
>>> s
[1, 2, 'three', (4, 5), [6, 7], 8, 'nine']

>>> s[0:2]=['one', 'two']
>>> s
['one', 'two', 'three', (4, 5), [6, 7], 8, 'nine']

>>> s.append(10)
>>> s
['one', 'two', 'three', (4, 5), [6, 7], 8, 'nine', 10]

>>> s.insert(5, 'four')
>>> s
['one', 'two', 'three', (4, 5), [6, 7], 'four', 8, 'nine', 10]

>>> s.remove('four')
>>> s
['one', 'two', 'three', (4, 5), [6, 7], 8, 'nine', 10]

>>> s.pop()
10

>>> s
['one', 'two', 'three', (4, 5), [6, 7], 8, 'nine']

>>> s.pop(0)
'one'

>>> s ['two', 'three', (4, 5), [6, 7], 8, 'nine']
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

List – Sort

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

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

```
>>> cars = ['Ford', 'BMW', 'Volvo']
>>> cars.sort()
>>> cars
['BMW', 'Ford', 'Volvo']

>>> cars.sort(reverse=True)
>>> cars
['Volvo', 'Ford', 'BMW']

>>> student = [("Nam", 10), ("Minh", 4), ("Xuan", 8), ("My", 2)]
>>> #How to sort by score?
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

List – Sort

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

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

```
>>> student = [("Nam", 10), ("Minh", 4), ("Xuan", 8), ("My", 2)]
>>> #How to sort by score?
>>> student.sort(key=lambda x: x[1])
>>> student
[('My', 2), ('Minh', 4), ('Xuan', 8), ('Nam', 10)]

>>> student.sort(key=lambda x: x[1], reverse=True)
>>> student
[('Nam', 10), ('Xuan', 8), ('Minh', 4), ('My', 2)]
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Basic String methods

Operation	Result
`center(width[, fillchar])`	Returns the string centered in a string of length width.
`count(sub[, start[, end]])`	Returns the number of non-overlapping occurrences of substring (sub) in the range [start, end].
`find(sub[, start[, end]])`	Returns the lowest index in the string where substring sub is found within the slice s[start:end]. Returns -1 if sub is not found.
`index(sub[, start[, end]])`	Like find() (above), but raises a ValueError when the substring is not found (find() returns -1 when the substring isn't found).
`replace(old, new[, count])`	Returns a copy of the string with all occurrences of substring old replaced by new.
`upper()`	Returns a copy of the string with all the cased characters converted to uppercase.
`split(sep=None, maxsplit=-1)`	Returns a list of the words in the string, using sep as the delimiter string.
`join(iterable)`	Returns a string which is the concatenation of the strings in iterable.

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Basic String methods

```
>>> demo = "pythonpp"
>>> demo.center(20,'-')
'------pythonpp------'

>>> demo.count('p')
3

>>> demo.find('o')
4

>>> demo.find('abc')
-1

>>> demo.index('o')
4

>>> demo.index('abc')
Traceback (most recent call last): File "<stdin>", line 1, in <module> ValueError: substring not
found

>>> demo.replace('p', '123')
'123ython123123'

>>> demo.upper()
'PYTHONPP'

>>> demo.split('o')
['pyth', 'npp']

>>> "-".join(("How", "are", "you", "?"))
'How-are-you-?'
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Basic String methods

Join & map

```
>>> a = [1,2,3,4,5]
>>> a
[1, 2, 3, 4, 5]
>>> "".join(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 0: expected str instance, int found
>>> "".join(map(str,a))
'12345'
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Set Types: set

- A set contains an unordered collection of unique objects.
- The set type is mutable — the contents can be changed using methods like `add()` and `remove()`
- If we want to create a set, we can call the built-in set function with a sequence or another iterable object:

```
>>> s = set("Hello world")
>>> s
{'o', 'r', 'e', 'H', 'd', ' ', 'w', 'l'}

>>> s2 = {1,2,3,4,1,2,3,4}
>>> s2
{1, 2, 3, 4}
```

- Set can't contain mutable object

```
>>> s3 = {1,2,3,4,(5,6)}
>>> s3
{1, 2, 3, 4, (5, 6)}

>>> s4 = {1,2,3,4,[5,6]}
Traceback (most recent call last): File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Set Operation

Operation	Result
`len(s)`	Return the number of elements in set s (cardinality of s).
`x in s`	Test x for membership in s.
`x not in s`	Test x for non-membership in s.
`isdisjoint(other)`	Return True if the set has no elements in common with other.
`issubset(other)`	Test whether the set is a proper subset of other

```
>>> s = {1,2,3,4,4,5}
>>> s
{1, 2, 3, 4, 5}

>>> 4 in s
True

>>> 3 not in s
False

>>> s.isdisjoint({5,6,7,8})
False

>>> s.isdisjoint({6,7,8})
True

>>> s.issubset({3,4,5,6,7})
False

>>> s.issubset({1,2,3,4,5,6,7})
True
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Set Operation

Operation	Result
`add(elem)`	Add element *elem* to the set.
`remove(elem)`	Remove element *elem* from the set. Raises KeyError if elem is not contained in the set.
`discard(elem)`	Remove element *elem* from the set if it is present.
`pop()`	Remove and return an arbitrary element from the set. Raises KeyError if the set is empty.
`clear()`	Remove all elements from the set.

```
>>> s = {1,2,3,4,5,6,6,6,7}
>>> s
{1, 2, 3, 4, 5, 6, 7}

>>> s.remove(5)
>>> s
{1, 2, 3, 4, 6, 7}

>>> s.remove(8)
Traceback (most recent call
last):
  File "<stdin>", line 1, in
<module>
KeyError: 8

>>> s.discard(8)
>>> s.discard(1)
>>> s
{2, 3, 4, 6, 7}

>>> s.pop()
2

>>> s
{3, 4, 6, 7}

>>> s.clear()
>>> s
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

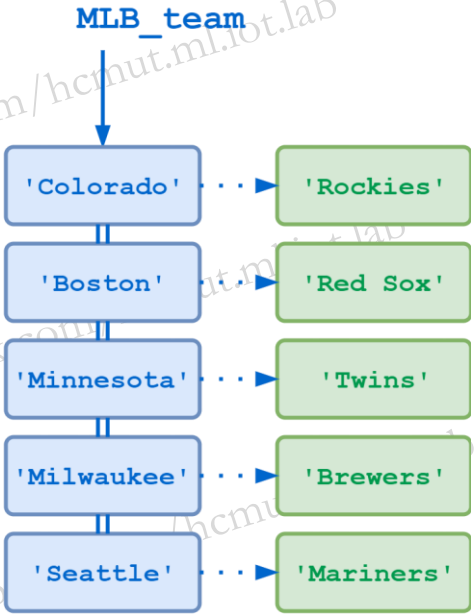
Mapping Types: dict

Dictionaries store a mapping between a set of keys and a set of values.

- Keys can be any immutable type.
- Values can be any type
- Values and keys can be of different types

You can

- define
- modify
- view
- lookup
- delete




```
d = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
```

```
>>> MLB_team = {
...     'Colorado' : 'Rockies',
...     'Boston'   : 'Red Sox',
...     'Minnesota': 'Twins',
...     'Milwaukee': 'Brewers',
...     'Seattle'  : 'Mariners'
... }
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Mapping Types: dict

- Keys must be unique.
- Assigning to an existing key replaces its value.
- Dictionaries are unordered
  - New entry might appear anywhere in the output

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == c == d == e
True
```

```
>>> d = {'user':'nkloi', 'pswd':'123456a@'}
>>> d
{'user': 'nkloi', 'pswd': '123456a@'}

>>> d['user']
'nkloi'

>>> d['id']=1
>>> d
{'user': 'nkloi', 'pswd': '123456a@', 'id': 1}

>>> d['pswd']="Abcd@1234"
>>> d
{'user': 'nkloi', 'pswd': 'Abcd@1234', 'id': 1}
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Dict Operations

Operation	Result
`get(key[, default])`	Return the value for key if key is in the dictionary, else default
`keys()`	Return a new view of the dictionary's keys.
`pop(key[, default])`	If key is in the dictionary, remove it and return its value, else return default.
`update([other])`	Update the dictionary with the key/value pairs from other, overwriting existing keys. Return *None*.
`clear()`	Remove all items from the dictionary.
`values()`	Return a new view of the dictionary's values.

```
>>> d = {'Name': 'Nguyen Van A', 'Salary': 400,
'City': 'Ho Chi Minh'}
>>> d
{'Name': 'Nguyen Van A', 'Salary': 400, 'City': 'Ho
Chi Minh'}

>>> d.get('City')
'Ho Chi Minh'

>>> d['City']
'Ho Chi Minh'

>>> d.keys()
dict_keys(['Name', 'Salary', 'City'])

>>> d.values()
dict_values(['Nguyen Van A', 400, 'Ho Chi Minh'])

>>> d.pop('Salary')
400

>>> d
{'Name': 'Nguyen Van A', 'City': 'Ho Chi Minh'}

>>> d2 = {'Name':'Nguyen Van B', 'Salary': 800}
>>> d2
{'Name': 'Nguyen Van B', 'Salary': 800}

>>> d.update(d2)
>>> d
{'Name': 'Nguyen Van B', 'City': 'Ho Chi Minh',
'Salary': 800}
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Dict Operations

Operation	Result
`items()`	Return a new view of the dictionary's items ((key, value) pairs)
`del d[key]`	Remove d[key] from d. Raises a KeyError if key is not in the map.
`key in d`	Return True if d has a key key, else False.
`key not in d`	Equivalent to not key in d.
`dict(zip(list_a, list_b))`	Dictionary from list

```
>>> list_a = ['ID', 'Name', 'Result']
>>> list_b = [142, 'NKLOI', 'Pass']
>>> d = dict(zip(list_a, list_b))
>>> d
{'ID': 142, 'Name': 'NKLOI', 'Result': 'Pass'}

>>> 'ID' in d
True

>>> del d['ID']
>>> d
{'Name': 'NKLOI', 'Result': 'Pass'}
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Output formatting

2 ways of writing values: print() and write() for file object, and also 2 ways to format an out.

- Handle by yourself, using string slicing and concatenation operations

```
>>> a = "Happy"
>>> b = "New Year"
>>> c = 2020
>>> print(a + ' ' + b, c)
Happy New Year 2020
```

- use str.format()

```
>>> a = "Happy"
>>> b = "New Year"
>>> c = 2020
>>> print('{} {} {}'.format(a,b,c))
Happy New Year 2020
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Output formatting

```
>>> print('{quantity} {item} cost ${price}'.format(
...     quantity=6,
...     item='bananas',
...     price=1.74))
6 bananas cost $1.74
```

```
print('{quantity} {item} cost ${price}'.format(
    quantity=6,
    item='bananas',
    price=1.74))
```

keyword_arguments

```
>>> print('{0} {1} cost ${2}'.format(6, 'bananas', 1.74))
6 bananas cost $1.74
```

```
print('{0} {1} cost ${2}'.format(6, 'bananas', 1.74))
```

template

positional_arguments

.format() method

```
6 bananas cost $1.74
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Positional Arguments

```
>>> '{0}/{1}/{2}'.format('foo', 'bar', 'baz')
'foo/bar/baz'

>>> '{2}.{1}.{0}/{0}{0}.{1}{1}.{2}{2}'.format('foo', 'bar',
'baz')
'baz.bar.foo/foofoo.barbar.bazbaz'

>>> '{}/{}/{}'.format('foo', 'bar', 'baz')
'foo/bar/baz'

>>> '{}{}'.format('foo', 'bar', 'baz')
'foobar'
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Output formatting

2 ways to write a table of squares and cubes

- str.rjust() – right justifies
- str.ljust(), str.center()

```
>>> for x in range(1,5):
...   print(str(x).rjust(2),
str(x**2).rjust(3),
str(x**3).rjust(3))
...
 1   1   1
 2   4   8
 3   9  27
 4  16  64
```

```
>>> for x in range(1,5):
...   print("{:2d} {:3d} {:3d}".format(x, x**2,
x**3))
...
 1   1   1
 2   4   8
 3   9  27
 4  16  64
```

String format

```
>>> print("{:03d} {:05.2f}".format(12, 12.5435345))
012 12.54
>>> print("{:*<10}".format("Hello"))
Hello*****
>>> print("{:*>10}".format("Hello"))
*****Hello
>>> print("{:>10}".format("Hello"))
     Hello
>>> print("{:^10}".format("Hello"))
  Hello
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Output formatting

Python 3.6+ added a new string formatting approach called formatted string literals or "f-strings". This new way of formatting strings lets you use embedded Python expressions inside string constants

```
>>> a = 10
>>> b = 15
>>> print(f'{a} + {b} = {a + b}')
10 + 15 = 25
>>> print('{a} + {b} = {a + b}')
{a} + {b} = {a + b}
```

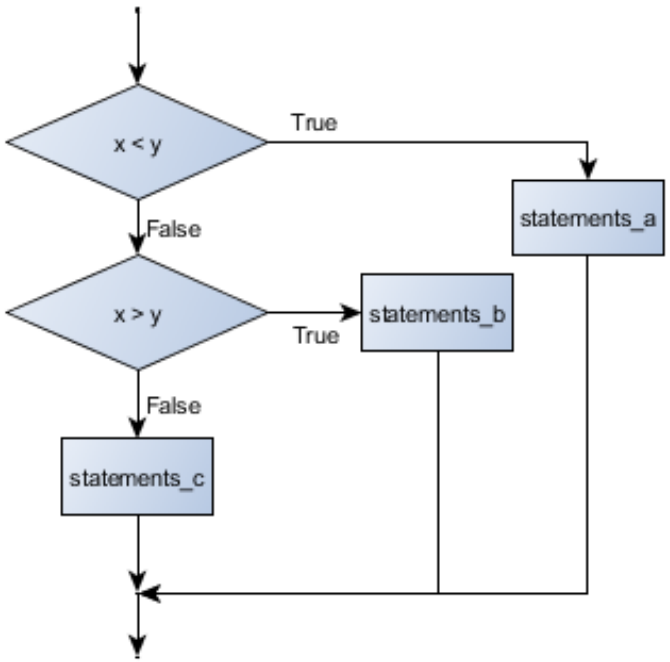
HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Conditions, branching & loops

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

The if else statement

```
if x < y:
    STATEMENTS_A
elif x > y:
    STATEMENTS_B
else:
    STATEMENTS_C
```




```
if choice == 'a':
    print("You chose 'a'.")
elif choice == 'b':
    print("You chose 'b'.")
elif choice == 'c':
    print("You chose 'c'.")
else:
    print("Invalid choice.")
```

```
<expr1> if <conditional_expr> else <expr2>
```

```
>>> m = a if a > b else b
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

The for loop

```
for LOOP_VARIABLE in SEQUENCE:
    STATEMENTS
```

```
for lover in ['Margot', 'Kathryn', 'Prisila']:
    mess = "I love you, " + lover
    print(mess)
```

```
>>> d = {'foo': 1, 'bar': 2, 'baz': 3}
>>> for k in d:
...     print(k)
...
foo
bar
Baz

>>> for k in d:
...     print(d[k])
...
1
2
3

>>> for v in d.values():
...     print(v)
...
1
2
3
```

```
for i in range(5):
    print(i)

Output:
0
1
2
3
4
```

```
>>> for (i, j) in (("Toan", 10), ("Sinh", 3), ("Ly", 4)):
...     print(i, ":", j)
...
Toan : 10
Sinh : 3
Ly : 4
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Enumerate

Don't use range() to iterate over a sequence solely to have the index and elements available at the same time

```
# Avoid (still work)
for i in range(len(mylist)):
    print ("{},{}".format(i, mylist[i]))
```

```
# Use
for (i, item) in enumerate(mylist):
    print ("{},{}".format(i, item))
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

While loop

```
while <expr>:
    <statement(s)>
```

```
>>> n = 5
>>> while n > 0:
...     n -= 1
...     print(n)
...
4
3
2
1
0
```

```
>>> a = ['foo', 'bar', 'baz']
>>> while a:
...     print(a.pop(-1))
...
baz
bar
foo
```

```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    continue
    <statement>
    <statement>
<statement>
```

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('Loop ended.')

----
4
3
Loop ended
```

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        continue
    print(n)
print('Loop ended.')

----
4
3
1
0
Loop ended.
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

List Comprehension

```
new_list = [expression for member in iterable]
```

```
>>> squares = [i * i for i in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
new_list = [expression for member in iterable (if conditional)]
```

```
>>> sentence = 'the rocket came back from mars'
>>> vowels = [i for i in sentence if i in 'aeiou']
>>> vowels
['e', 'o', 'e', 'a', 'e', 'a', 'o', 'a']
```

```
new_list = [expression (if conditional) for member in iterable]
```

```
>>> original_prices = [1.25, -9.45, 10.22, 3.78, -5.92, 1.16]
>>> prices = [i if i > 0 else 0 for i in original_prices]
>>> prices
[1.25, 0, 10.22, 3.78, 0, 1.16]
```

Functions in Python

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Functions in Python

```
def function_name([<parameters>]):
    <statement(s)>
    return [<values>]
```

```
def f(qty, item, price):
    print(f'{qty} {item} cost ${price:.2f}')

>>> f(6, 'bananas', 1.74)
6 bananas cost $1.74
```

Function Call

```
f(6, 'bananas', 1.74)
```

arguments
(actual parameters)

→

Function Definition

```
def f(qty, item, price):
```

parameters
(formal parameters)

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Default Parameters

```
def f(qty=6, item='bananas', price=1.74):
    print(f'{qty} {item} cost ${price:.2f}')
```

```
>>> f(4, 'apples', 2.24)
4 apples cost $2.24
>>> f(4, 'apples')
4 apples cost $1.74

>>> f(4)
4 bananas cost $1.74
>>> f()
6 bananas cost $1.74

>>> f(item='kumquats', qty=9)
9 kumquats cost $1.74
>>> f(price=2.29)
6 bananas cost $2.29
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Variable-Length Argument Lists

```
def avg(a, b=0, c=0, d=0, e=0):
    # Calculate average
    pass

avg(1)
avg(1, 2)
avg(1, 2, 3)
avg(1, 2, 3, 4)
avg(1, 2, 3, 4, 5)
```

```
def avg(a, b=0, c=0, d=0, e=0):
    return (a + b + c + d + e) / # Divided by what???
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Variable-Length Argument Lists

```
>>> def avg(a):
...     total = 0
...     for v in a:
...         total += v
...     return total / len(a)
...

>>> avg([1, 2, 3])
2.0

>>> avg([1, 2, 3, 4, 5])
3.0

>>> t = (1, 2, 3, 4, 5)
>>> avg(t)
3.0
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Argument Tuple Packing

```
>>> def f(*args):
...     print(args)
...     print(type(args), len(args))
...     for x in args:
...             print(x)
...
>>> f(1, 2, 3)
(1, 2, 3)
<class 'tuple'> 3
1
2
3

>>> f('foo', 'bar', 'baz', 'qux', 'quux')
('foo', 'bar', 'baz', 'qux', 'quux')
<class 'tuple'> 5
foo
bar
baz
qux
quux
```

Argument Tuple Packing

```
def avg(a, b=0, c=0, d=0, e=0):
        return (a + b + c + d + e) / # Divided by??
        pass

avg(1)
avg(1, 2)
avg(1, 2, 3)
avg(1, 2, 3, 4)
avg(1, 2, 3, 4, 5)
```

```
>>> def avg(*args):
...     total = 0
...     for i in args:
...         total += i
...     return total / len(args)
...

>>> avg(1, 2, 3)
2.0

>>> avg(1, 2, 3, 4, 5)
3.0
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Argument Dictionary Packing

```
>>> def f(**kwargs):
...     print(kwargs)
...     print(type(kwargs))
...     for key, val in kwargs.items():
...             print(key, '->', val)
...

>>> f(foo=1, bar=2, baz=3)
{'foo': 1, 'bar': 2, 'baz': 3}
<class 'dict'>
foo -> 1
bar -> 2
baz -> 3
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab

Putting It All Together

```
>>> def f(a, b, *args, **kwargs):
...     print(F'a = {a}')
...     print(F'b = {b}')
...     print(F'args = {args}')
...     print(F'kwargs = {kwargs}')
...

>>> f(1, 2, 'foo', 'bar', 'baz', 'qux', x=100, y=200, z=300)
a = 1
b = 2
args = ('foo', 'bar', 'baz', 'qux')
kwargs = {'x': 100, 'y': 200, 'z': 300}
```

HCMUT EE Machine Learning & IoT Lab https://www.facebook.com/hcmut.ml.iot.lab