ภาควิชาคณิตศาสตร์, สาขาคอมพิวเตอร์
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

Python Programming

v. 1.1.0

ดร. พีระจักร์ วิฑูรชาติ
peerajak.wito@kmutt.ac.th

ขอบคุณ latex template บีมเมอร์ธีมขนนก โดย B. Intiyot

January 26, 2022

สารบัญ 1

object คืออะไร

Class Definition

```
class ComplexNumber:
    def __init__(self, r=0, i=0):
        self.real = r
        self.imag = i

    def get_data(self):
        print(f'{self.real}+{self.imag}j')
```

Object initialization, and deletion

```
>>> c1 = ComplexNumber(1,3)
>>> c1.get_data()
1+3j
>>> del c1
```

objectคืออะไร

- Object คือ Instance ของ Class. วัตถุ คือ ชิ้นตัวอย่าง ของ ประเภท
- ยกตัวอย่างเช่น Class ComplexNumber คือแนวคิด หรือพิมพ์เขียวที่บอกว่า ComplexNumber มีลักษณะอย่างไร มีอะไรบ้าง กลไกข้างในทำอย่างไร
- Functions ใน Class เราเรียกว่า methods
- Variables ใน Class เราเรียกว่า attributes
- Instance ของ Class มีการเกิดและตาย เมื่อเราสร้างinstance ก่อให้เกิดการจองหน่วยความจำ และเมื่อเราเลิกใช้instance หน่วยความจำก็จะถูกลบออกไปจากหน่วยความจำ
- การที่ชื่อตัวแปรชี้ไปที่ instance เราเรียกว่าreference

What is an object 2

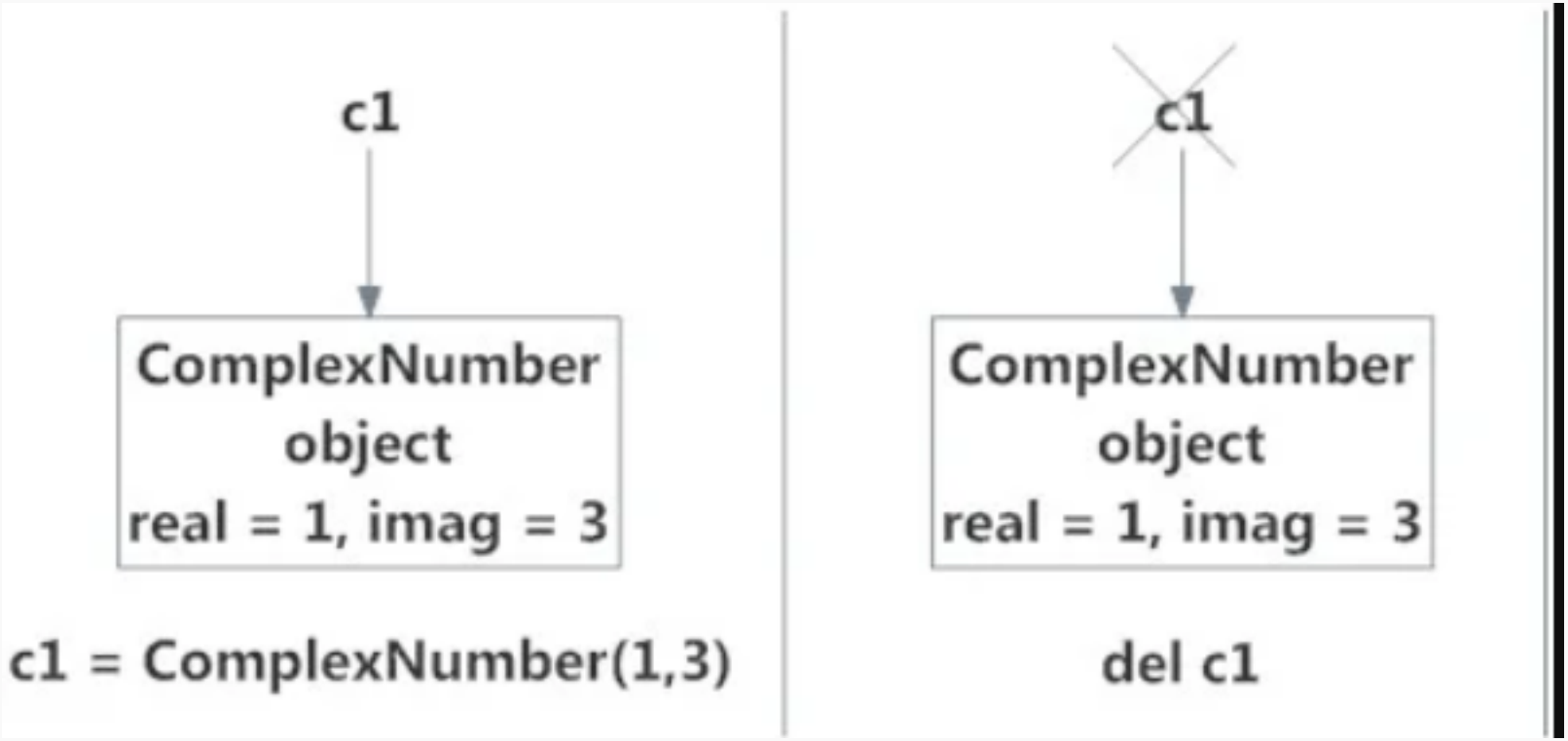

Figure: Object creation, and deletion

Everything is an object


```
>>> age=19
>>> id(age)
9756800
>>> type(age)
<class 'int'>
>>> age
19
```

Mutable or Immutable

- Object ที่สามารถเปลี่ยนค่าได้ เรียกว่า Mutable object
- Object ที่ไม่สามารถเปลี่ยนค่าได้ เรียกว่า Immutable object

Mutable or Immutable

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)	

Figure: Class and mutability

Mutable or Immutable2

Check with id()

```
>>> age = 42
>>> id(age)
4377553168
>>> age = 43
>>> id(age)
4377553200
```

age เปลี่ยนการ reference ไป object อื่น แบบนี้แสดงว่า integerเป็น immutbale

Mutable or Immutable

- Object จาก Class ที่เราสร้างเองเป็น Mutable object
- เมื่อเราเขียน age=42,objectใหม่เกิดขึ้นจากนั้นเกิดการโยงชื่อageไปที่objectใหม่ เมื่อเราเขียนage=43 objectใหม่อีกอันก็เกิดขึ้นจากนั้นเกิดการโยงชื่อageไปที่objectใหม่
- ฟังก์ชั่นชื่อว่า id() เรียกว่า built-in function คือฟังก์ชั่นที่แถมมากับpython interpreter
- Built-in Function: https://docs.python.org/3/library/functions.html

Mutable or Immutable3


```
>>> class Person():
...  def __init__(self, age):
...  self.age = age
...
>>> fab = Person(age=42)
>>> fab.age
42
>>> id(fab)
4380878496
>>> id(fab.age)
4377553168
>>> fab.age = 25 # I wish!
>>> id(fab) # will be the same
4380878496
>>> id(fab.age) # will be different
4377552624
```

Number


```
>>> a = 14
>>> b = 3
>>> a + b # addition
17
>>> a - b # subtraction
11
>>> a * b # multiplication
42
>>> a / b # true division
4.666666666666667
>>> a // b # integer division
4
>>> a % b # modulo operation (reminder of division)
2
>>> a ** b # power operation
2744
```

Complex Number


```
>>> c = 3.14 + 2.73j
>>> c.real # real part
3.14
>>> c.imag # imaginary part
2.73
>>> c.conjugate() # conjugate of A + Bj is A - Bj
(3.14-2.73j)
>>> c * 2 # multiplication is allowed
(6.28+5.46j)
>>> c ** 2 # power operation as well
(2.4067000000000007+17.1444j)
>>> d = 1 + 1j # addition and subtraction as well
>>> c - d
(2.14+1.73j)
```

Fractions


```
>>> from fractions import Fraction
>>> Fraction(10, 6) # mad hatter?
Fraction(5, 3) # notice it's been simplified
>>> Fraction(1, 3) + Fraction(2, 3) # 1/3 + 2/3 == 3/3 == 1/1
Fraction(1, 1)
>>> f = Fraction(10, 6)
>>> f.numerator
5
>>> f.denominator
3
```

Strings


```
>>> # 4 ways to make a string
>>> str1 = 'This is a string. We built it with single quotes.'
>>> str2 = "This is also a string, but built with double quotes."
>>> str3 = '''This is built using triple quotes,
... so it can span multiple lines.'''
>>> str4 = """This too
... is a multiline one
... built with triple double-quotes."""
>>> str4 #A
'This too\nis a multiline one\nbuilt with triple double-quotes.'
>>> print(str4) #B
This too
is a multiline one
built with triple double-quotes.
>>> len(str1)
49
```

Unicode


```
>>> s = "This is ŋüícode ภาษาไทย " # unicode string: code points
>>> type(s)
<class 'str'>
>>> encoded_s = s.encode('utf-8') # utf-8 encoded version of s
>>> encoded_s
b'This is \xc3\xbc\xc5\x8b\xc3\xadcode' # result: bytes object
>>> type(encoded_s) # another way to verify it
<class 'bytes'>
>>> encoded_s.decode('utf-8') # let's revert to the original
'This is ŋüícode ภาษาไทย '
>>> bytes_obj = b"A bytes object" # a bytes object
>>> type(bytes_obj)
<class 'bytes'>
```

String indexing,Slicing


```
>>> s = "The trouble is you think you have time."
>>> s[0] # indexing at position 0, which is the first char
'T'
>>> s[5] # indexing at position 5, which is the sixth char
'r'
>>> s[:4] # slicing, we specify only the stop position
'The '
>>> s[4:] # slicing, we specify only the start position
'trouble is you think you have time.'
>>> s[2:14] # slicing, both start and stop positions
'e trouble is'
>>> s[2:14:3] # slicing, start, stop and step (every 3 chars)
'erb '
>>> s[:] # quick way of making a copy
'The trouble is you think you have time.'
```

String Formatting


```
>>> greet_old = 'Hello %s!'
>>> greet_old % 'Fabrizio'
'Hello Fabrizio!'

>>> greet_positional = 'Hello {} {}!'
>>> greet_positional.format('Fabrizio', 'Romano')
'Hello Fabrizio Romano!'

>>> greet_positional_idx = 'This is {0}! {1} loves {0}!'
>>> greet_positional_idx.format('Python', 'Fabrizio')
'This is Python! Fabrizio loves Python!'
>>> greet_positional_idx.format('Coffee', 'Fab')
'This is Coffee! Fab loves Coffee!'

>>> keyword = 'Hello, my name is {name} {last_name}'
>>> keyword.format(name='Fabrizio', last_name='Romano')
'Hello, my name is Fabrizio Romano'
```

String Formatting: f-string


```
>>> name = 'Fab'
>>> age = 42
>>> f"Hello! My name is {name} and I'm {age}"
"Hello! My name is Fab and I'm 42"
>>> from math import pi
>>> f"No arguing with {pi}, it's irrational..."
"No arguing with 3.141592653589793, it's irrational..."
>>> a = 5
>>> b = 10
>>> f'Five plus ten is {a + b} and not {2 * (a + b)}.'
'Five plus ten is 15 and not 30.'
```

อันนี้เรียกว่า f-string

Tuple


```
>>> t = () # empty tuple
>>> type(t)
<class 'tuple'>
>>> one_element_tuple = (42, ) # you need the comma!
>>> three_elements_tuple = (1, 3, 5) # braces are optional her
>>> a, b, c = 1, 2, 3 # tuple for multiple assignment
>>> a, b, c # implicit tuple to print with one instruction
(1, 2, 3)
>>> 3 in three_elements_tuple # membership test
True
```

Tuple swap


```
>>> a, b = 0, 1
>>> a, b = b, a # this is the Pythonic way to do it
>>> a, b
(1, 0)
```

Unpacking tuples


```
>>> record = ('Dave', 'dave@example.com',
 '773-555-1212', '847-555-1212')
>>> type(record)
<class 'tuple'>
>>> name, email, *phone_numbers = record
>>> phone_numbers
['773-555-1212', '847-555-1212']
>>> type(phone_numbers)
<class 'list'>
```

phone_numbers เป็นชนิด List

Mutable Sequences

- List, Dict, Set, Enum ล้วนเป็น mutuable sequences
- แม้ว่า Enums are mutbale, สมาชิกของมันก็ไม่สามารถเปลี่ยนแปลงได้

Lists


```
>>> [] # empty list
[]
>>> list() # same as []
[]
>>> [1, 2, 3] # as with tuples, items are comma sep
[1, 2, 3]
>>> [x + 5 for x in [2, 3, 4]] # Python is magic
[7, 8, 9]
>>> list((1, 3, 5, 7, 9)) # list from a tuple
[1, 3, 5, 7, 9]
>>> list('hello') # list from a string
['h', 'e', 'l', 'l', 'o']
```

Lists


```
>>> a = [1, 2, 1, 3]
>>> a.append(13)  # we can append anything at the end
>>> a
[1, 2, 1, 3, 13]
>>> a.count(1)  # how many '1' are there in the list?
2
>>> a.extend([5, 7])  # extend the list by another (or sequence)
>>> a
[1, 2, 1, 3, 13, 5, 7]
>>> a.index(13)  # position of '13' in the list (0-based indexing)
4
>>> a.insert(0, 17)  # insert '17' at position 0
>>> a
[17, 1, 2, 1, 3, 13, 5, 7]
>>> a.pop()  # pop (remove and return) last element
7
>>> a.pop(3)  # pop element at position 3
1
>>> a
[17, 1, 2, 3, 13, 5]
>>> a.remove(17)  # remove '17' from the list
>>> a
```

Packing Lists


```
>>> keys =['one','two','three']
>>> values =[1.,2.,3.]
>>> keyvalues = [keys,values]
>>> keyvalues
[['one', 'two', 'three'], [1.0, 2.0, 3.0]]
>>> keyvalues = [*keys,*values]
>>> keyvalues
['one', 'two', 'three', 1.0, 2.0, 3.0]
```

Unpacking List


```
>>> records
[('foo', 1, 2), ('bar', 'hello'), ('foo', 3, 4)]
>>> def do_foo(x,y):
...     print('foo',x,y)
...
>>> def do_bar(s):
...     print('bar',s)
...
>>> for tag , *args in records:
...     if tag == 'foo':
...         do_foo(*args)
...     elif tag == 'bar':
...         do_bar(*args)
...
foo 1 2
bar hello
foo 3 4
```

Unpacking List2


```
>>> for tag, *args in records:
...         print(tag,args)
...         type(args)
...
foo [1, 2]
<class 'list'>
bar ['hello']
<class 'list'>
foo [3, 4]
<class 'list'>
```

args เป็นคลาสลิสต์

Chapter02/ch2/sets.py


```
# sets.py
>>> small_primes = set()  # empty set
>>> small_primes.add(2)  # adding one element at a time
>>> small_primes.add(3)
>>> small_primes.add(5)
>>> small_primes
{2, 3, 5}
>>> small_primes.add(1)  # Look what I've done, 1 is not a prin
>>> small_primes
{1, 2, 3, 5}
>>> small_primes.remove(1)  # so let's remove it
>>> 3 in small_primes  # membership test
True
>>> 4 in small_primes
False
>>> 4 not in small_primes  # negated membership test
True
```

Chapter02/ch2/dicts.py


```
>>> d=dict(a=1,b=2)
>>> d
{'a': 1, 'b': 2}
>>> d={'a':1,'b':2}
>>> d
{'a': 1, 'b': 2}
>>> len(d)  # how many pairs?
2
>>> d['a']  # what is the value of 'a'?
1
>>> d  # how does ‘d‘ look now?
{'a': 1, 'b': 2}
>>> del d['a']  # let's remove ‘a‘
>>> d
{'b': 2}
>>> d['c'] = 3  # let's add 'c': 3
>>> 'c' in d  # membership is checked against the keys
True
>>> 3 in d  # not the values
```

Chapter02/ch2/dicts.py 2


```
# views
>>> d = dict(zip('hello', range(5)))
>>> d
{'h': 0, 'e': 1, 'l': 3, 'o': 4}
>>> d.keys()
dict_keys(['h', 'e', 'l', 'o'])
>>> d.values()
dict_values([0, 1, 3, 4])
>>> d.items()
dict_items([('h', 0), ('e', 1), ('l', 3), ('o', 4)])
>>> 3 in d.values()
True
>>> ('o', 4) in d.items()
True
# other methods
>>> d
{'e': 1, 'h': 0, 'o': 4, 'l': 3}
```

Chapter02/ch2/dicts.py 3


```
>>> d.popitem()  # removes a random item (useful in algorithms
('o', 4)
>>> d
{'h': 0, 'e': 1, 'l': 3}
>>> d.pop('l')  # remove item with key 'l'
3
>>> d.pop('not-a-key')  # remove a key not in dictionary: KeyE
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'not-a-key'
>>> d.pop('not-a-key', 'default-value')
# with a default value?
'default-value'  # we get the default value
```

Chapter02/ch2/dicts.py 4


```
>>> d.update({'another': 'value'})
# we can update dict this way
>>> d.update(a=13)  # or this way (like a function call)
>>> d
{'h': 0, 'e': 1, 'another': 'value', 'a': 13}
>>> d.get('a')  # same as d['a'] but if key is missing no KeyE
13
>>> d.get('a', 177)  # default value used if key is missing
13
>>> d.get('b', 177)  # like in this case
177
>>> d.get('b')  # key is not there, so None is returned
```

Chapter02/ch2/dicts.py 5


```
# setdefault
>>> d = {}
>>> d.setdefault('a', 1)  # 'a' is missing, we get defau
1
>>> d
{'a': 1}  # also, the key/value pair ('a', 1) has now be
>>> d.setdefault('a', 5)  # let's try to override the va
1
>>> d
{'a': 1}  # no override, as expected
# setdefault example
>>> d = {}
>>> d.setdefault('a', {}).setdefault('b', []).append(1)
>>> d
{'a': {'b': [1]}}
```

Packing a Dict from two lists


```
>>> keys =['one','two','three']
>>> values =[1.,2.,3.]
>>> dictKeyValues = dict(zip(keys,values))
>>> dictKeyValues
{'one': 1.0, 'two': 2.0, 'three': 3.0}
```

Chapter02/ch2/enum.py


```
>>> GREEN = 1
>>> YELLOW = 2
>>> RED = 4
>>> TRAFFIC_LIGHTS = (GREEN, YELLOW, RED)
>>> # or with a dict
>>> traffic_lights = {'GREEN': 1, 'YELLOW': 2, 'RED': 4}
# using enum
>>> from enum import Enum
>>> class TrafficLight(Enum):
...     GREEN = 1
...     YELLOW = 2
...     RED = 4
...
>>> TrafficLight.GREEN
<TrafficLight.GREEN: 1>
>>> TrafficLight.GREEN.name
'GREEN'
```

Chapter02/ch2/enum.py


```
>>> TrafficLight.GREEN.value
1
>>> TrafficLight(1)
<TrafficLight.GREEN: 1>
>>> TrafficLight(4)
<TrafficLight.RED: 4>
```

Chapter02/ch2/final considerations 1.py


```
# final_considerations.py
>>> a = 1000000
>>> b = 1000000
>>> id(a) == id(b)
False
>>> a = 5
>>> b = 5
>>> id(a) == id(b)
True
```

อันนี้เพราะว่า Python จำ object ตัวเลขที่ใช้บ่อยๆเอาไว้

Chapter02/ch2/final considerations 2.py


```
# how to choose data structures
# example customer objects
customer1 = {'id': 'abc123', 'full_name': 'Master Yoda'}
customer2 = {'id': 'def456', 'full_name': 'Obi-Wan Kenob
customer3 = {'id': 'ghi789', 'full_name': 'Anakin Skywal
# collect them in a tuple
customers = (customer1, customer2, customer3)
# or collect them in a list
customers = [customer1, customer2, customer3]
# or maybe within a dictionary, they have a unique id af
customers = {
    'abc123': customer1,
    'def456': customer2,
    'ghi789': customer3,
}
```

การเลือกobjectสำหรับเก็บข้อมูล

- ปกติเราจะไม่ใช้ tuple เก็บข้อมูล เพราะมัน immutable ยกเว้นต้องการจะบอกว่าข้อมูลเหล่านี้จะไม่เปลี่ยนแปลง
- ปกติมักจะใช้ลิสต์ เพราะว่ามันทำงานด้วยง่าย
- แต่ถ้าคุณต้องการให้มันไม่มีก่อนหลัง ควรใช้set.
- list อนุญาตให้เก็บข้อมูลเดิมหลายครั้งได้ แต่set ไม่อนุญาต
- set สามารถทำ set operation แบบคณิตศาสตร์ได้ เช่น union, intersection

```
>>> aset
{'d', 1, 3, (1, 2, 3)}
>>> aset.add(1)
>>> aset
{'d', 1, 3, (1, 2, 3)}
```

การเลือกobjectสำหรับเก็บข้อมูล

- สมมุติจะหาของในลิสต์ จะต้องไล่หาทีละตัวว่าใช่หรือไม่ ใช่หรือไม่ ใช่หรือไม่ จนเจอ
- การกระทำนี้ใช้จำนวนครั้งเท่ากับจำนวนข้อมูลในลิสต์
- ถ้าลิสต์เก็บสิบล้านบันทึก จะใช้เวลาหานานมาก
- แต่ถ้า dict จะหาเจอทันที ครั้งเดียว เพราะว่าkey ของdict มันต้องunique