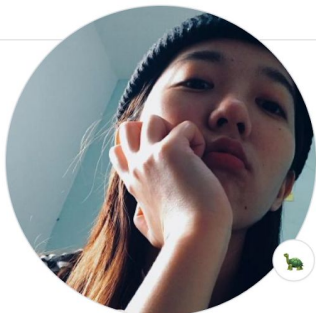


Python Tutorial

NCHU Lab.5501

2020.10.23 Joy Liao

About Me



Joy Liao
plusoneee

Edit profile

7 followers · 4 following · ☆ 54

Taiwan

y02121148s@gmail.com

Highlights

* Arctic Code Vault Contributor

Organizations



Overview

Repositories 24

Projects

Packages

plusoneee / README.md

Send feedback



Work Experience

Backend Engineer at SELLERLINX

Jul 2019 - Aug 2020, Taipei City, Taiwan.

- We build Serverless Application on Amazon Web Service (AWS).
- My main job was to build RESTful APIs using Lambda, API Gateway, DynamoDB, S3, RDS (MySQL), SQS, etc. Besides, I further had experience in creating microservices using ECR and ECS during this period.

Pinned Order updated.

Customize your pins

music.cluster

Includes two parts, the first part for automatically create feature datasets from Spotify playlists, another for music clusters.

Python ☆ 1 🍴 1

music.platform

As an experimental music platform, be used to collect users' listening data.

TSQL

_nchuzero

Record some of my practice about the ML & DL.

Jupyter Notebook

crawl.big.TW

Python ☆ 1 🍴 1



01

BASIC REVIEW

- Condition
- Loop
- Function

03

CLASS INHERITANCE

- Method Overriding
- Multi-Level Inheritance

02

CLASS & INSTANCE

- Attribute
- Static/Class Method

04

WELL-MAINTAINED CODE

- Cohesion & Coupling
- Clean Code Tips

01

BASIC REVIEW-1

```
2   # Define a variable in Python
3   my_name = 'Joy'
4   myName = 'Joy'
5   IAM_RELLY_COOL = True
6
7   # How to Define a function
8   def do_something_really_cool_like_joy():
9       # You could do something in the function.
10      # for example, like print string.
11      print('OK, Maybe ...')
12      print('Study Machine Learning hard ? ')
13      # ok, maybe this example not cool.
14
15  def doNothing():
16      # and you also could return something in the end of the function.
17      # here return None bcz the function name call `do nothing` haha.
18      return None
19
20  # If condition
21  if IAM_RELLY_COOL:
22      do_something_really_cool_like_joy()
23
24  elif not IAM_RELLY_COOL:
25      do_what = doNothing()
26
27  else:
28      pass
```

01

BASIC REVIEW-2

```
30 # For-loop a range
31 for i in range(10):
32     print('No.', i, 'time in this for-loop.')
33
34 # For-loop a `list`
35 artist_joy_like = ['deca joins', '9m88', 'Leo Wang', 'Ryan Beatty',
36                   'MAMAMOO', 'Soft Lipa', 'Waa Wei', 'The Black Skirts']
37
38 # okay, so if you also like these singer, contact me. (X)
39 for artist in artist_joy_like:
40     print(artist)
41
42 # For-loop in one-line
43 odd_numbers = [n for n in range(20) if n % 2 == 1]
44 print(odd_numbers)
45
46 # Equal to
47 odd_numbers = []
48 for n in range(20):
49     if n % 2 == 1:
50         odd_numbers.append(n)
51 print(odd_numbers)
```

02

CLASS & INSTANCE

- Define a Person Class.
- Define a Function in Class.
- Instance it !!!

```
2 # How to define a class
3 # A Class is like an object constructor,
4 # or a "blueprint" for creating objects.
5
6 class Person:
7     def __init__(self, name):
8         self.name = name # necessary attribute
9         self.birthday = None # not necessary
10
11     def say_hi(self):
12         # method
13         print('%s say HI!' % self.name)
14
15 # instance it
16 joy = Person('Joy')
17
18 # now, we have an Person object.
19 # and this object is able to say hi.
20 joy.say_hi()
21
22 # We don't know joy's birthday.
23 if joy.birthday is None:
24     print('We don\'t know about the birthday.')
25     joy.birthday = input('Could you tell us? ')
26
27 print('Okay,', joy.name, '\',s birthday at', joy.birthday, '.')
```

“A soulmate is a person with whom one has a feeling of deep or natural affinity.
This may involve similarity, love, comfort and trust.”

— “Soulmate” definition from Wikipedia.

Static Method

- Write another class: Soulmate.
- A static method is also a method which is bound to the class and not the object of the class.

```
2
3 class Soulmate():
4     def __init__(self, name, in_common=[]):
5         self.name = name
6         self.birthday = None
7         self.in_common = in_common
8
9     def list_in_common(self):
10        print('%s and you have many things in common, like:' % self.name)
11        for item in self.in_common:
12            print(item)
13        return self.in_common
14
15    @staticmethod
16    def definition():
17        print('Definition of Soulmate: ')
18        print('A soulmate is a person with whom one has a feeling of deep or natural affinity.'
19              ' This may involve similarity, love, comfort and trust.')
20
21
22 # static method no need to instantiate the object.
23 # for example: definition()
24 Soulmate.definition()
25
26 # list_in_common need to.
27 byul = Soulmate('byul', ['foods', 'cool', 'experience'])
28 byul.list_in_common()
```


Class Method

- Write another class: Lover.
- A class methods can be called by both class and object.

```
2
3 class Lover():
4     weekends_with_you = 'My lover lover lover don\'t say no.\nI just wanna head home I don\'t feel so well.'
5
6
7     def __init__(self, name, date_from='Someday'):
8         self.name = name
9         self.date_from = date_from
10        self.memories = list()
11
12        @classmethod
13        def do_something(cls):
14            print(cls.weekends_with_you)
15            return cls.weekends_with_you
16
17        def add_memory(self, memory):
18            self.memories.append(memory)
19
20        def recall_memory(self):
21            print(self.memories)
22            return self.memories
23
24        # call classmethod without instance it.
25        Lover.do_something()
26
27        # call classmethod by object.
28        someone_i_love = Lover('JC', '2020-10-20')
29        someone_i_love.do_something()
```

03

INHERITANCE

- Inheritance allows us to define a class that inherits all the methods and properties from another class.

```
2  # Import a module
3  # from <File Name> import <Class Name> or <Function Name>
4  from base_person import Person
5
6  # Bcz the definition of SoulMate "soulmate is a person".
7  # Soulmate will inherit the properties and methods from the People.
8
9  class SoulMate(Person):
10     def __init__(self, name, in_common):
11         # By using the super() function,
12         # you do not have to use the name of the parent element,
13         # it will automatically inherit the methods and properties from its parent
14
15         super(SoulMate, self).__init__(name)
16         self.in_common = in_common
17
18     def list_in_common(self):
19         print('%s and you have many things in common, like:' % self.name)
20         for item in self.in_common:
21             print(item)
22         return self.in_common
23
24     @staticmethod
25     def definition():
26         print('Definition of Soulmate: ')
27         print('A soulmate is a person with whom one has a feeling of deep or natural affinity.'
28               ' This may involve similarity, love, comfort and trust.')
29
30
31
32 byul = SoulMate('byul', ['cool', 'experience', 'tattoos'])
33 byul.list_in_common()
34
35 # inherit the methods 'say_hi()' from the Person.
36 byul.say_hi()
37
38 # inherit the attribute 'birthday' from the Person
39 byul.birthday = '1997-12-20'
40 print(byul.birthday)
```

MULTI-LEVEL INHERITANCE

- In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.
- Method overriding occurs by simply defining in the child class a method with the same name of a method in the parent class.

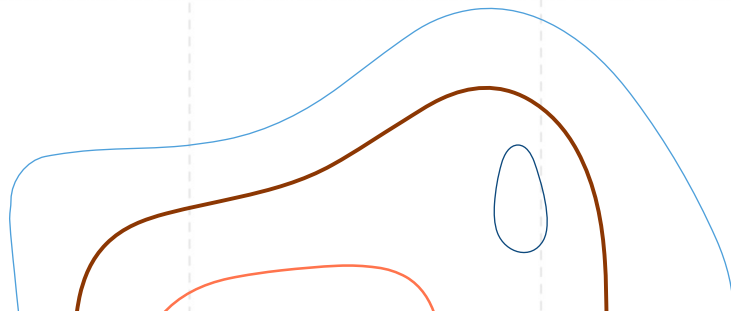
```
2  # Import a module
3  # from <File Name> import <Class Name> or <Function Name>
4  from soulmate import SoulMate
5
6  class Lover(SoulMate):
7      weekends_with_you = 'My lover lover lover don\'t say no.\nI just wanna head home I don\'t feel so well.'
8
9
10 def __init__(self, name, in_common, date_from):
11     super(Lover, self).__init__(name, in_common)
12     self.date_from = date_from
13     self.memories = list()
14
15 def say_hi(self):
16     print('%s say good morning.' % self.name)
17
18 @classmethod
19 def do_something(cls):
20     print(cls.weekends_with_you)
21
22 def add_memory(self, memory):
23     self.memories.append(memory)
24
25 def recall_memory(self):
26     print(self.memories)
27     return self.memories
28
29 jian = Lover('Jian', ['music'], '2020-10-20')
30
31 # inherit the function 'list_in_common()' from the SoulMate.
32 jian.list_in_common()
33
34 # could replace the function 'say_hi()' from People.
35 jian.say_hi()
```



04

Well-MAINTAINED CODE

- Cohesion & Coupling
- Define PATH in a good way



COHESION

Refers to what the class or module can do.

- Low Cohesion:
 - The class does a great variety of actions.
 - Unfocused on what it should do.
- High Cohesion:
 - The class is focused on what it should be doing.
 - Only methods relating the intention of the class.

COUPLING

How related or dependent two class/modules are toward each other.

- **Low Coupled:**
 - Change something major in one class should not affect the other.
- **High Coupled:**
 - Make it difficult to change and maintain your code.
 - Making a change that could require an entire system revamp.



High Cohesion

Low Coupling

Okay, Let me give some examples ...

Example 01

```
4 train_df = pd.read_csv("./a_random_dataset.csv")
5 for i in range(len(train_df['a_specific_col'])):
6     str = train_df.loc[i, 'a_specific_col']
7
8     if str == 'a_value':
9         train_df.loc[i, 'a_specific_col'] = 0
10
11     elif str == 'b_value':
12         train_df.loc[i, 'a_specific_col'] = 1
13
14     elif str == 'c_value':
15         train_df.loc[i, 'a_specific_col'] = 2
16
17     elif str == 'd_value':
18         train_df.loc[i, 'a_specific_col'] = 3
19
20     elif str == 'e_value':
21         train_df.loc[i, 'a_specific_col'] = 4
```

Think for a while

Is a good example or a bad example ?

But, WHY ?

Example 01

```
24 def tokenizer_by_column(df, col_name=None):
25     values_unique_list = df[col_name].unique()
26     count = 0
27     for value in values_unique_list:
28         mask = (df[col_name] == value)
29         df[col_name][mask] = count
30         count += 1
31     return df
32
33
34 train_df = pd.read_csv("./a_random_dataset.csv")
35 train_df = tokenizer_by_column(train_df, 'a_specific_col')
```

Okay, seems like
BETTER ?

Example 02

Here is the code from my ML homework haha.

```
8 class Standardize:
9
10     @staticmethod
11     def robust_scaler(df, col_name=None, q_range=(25, 75)):...
12
13
14     @staticmethod
15     def normailizer(df, col_name=None, method='l2'):...
16
17
18     @staticmethod
19     def standard_scaler(df, col_name=None):...
20
21
22 class ValueReplacer:
23
24     @staticmethod
25     def replace_value(df, col_name, new_value, old_value):...
26
27
28     @staticmethod
29     def tokenizer_by_column(df, col_name=None):...
30
31
32     @staticmethod
33     def fill_missing_value(df, col_name=None, method='mean'):...
34
35
36     @staticmethod
37     def fill_missing_value_by_imputer(df, col_name=None, method='mean'):...
```

Think for a while

Is a good example or a bad example ?

But, WHY ?

Example 02

That why we need function and module !

```
12 from preprocess import ValueReplacer, Standardize
13 def pipeline():
14     bank = pd.read_csv('bank/bank-full.csv', delimiter=';')
15     bank = ValueReplacer.tokenizer_by_column(bank, 'housing')
16     bank = ValueReplacer.tokenizer_by_column(bank, 'loan')
17     bank = ValueReplacer.tokenizer_by_column(bank, 'default')
18     bank = ValueReplacer.tokenizer_by_column(bank, 'marital')
19     bank = ValueReplacer.tokenizer_by_column(bank, 'job')
20     bank = ValueReplacer.tokenizer_by_column(bank, 'education')
21
22     bank = ValueReplacer.replace_value(bank, 'poutcome', 'failure', 'other')
23     bank = ValueReplacer.replace_value(bank, 'poutcome', 'failure', 'unknown')
24     bank = ValueReplacer.tokenizer_by_column(bank, 'poutcome')
25
26     bank = Standardize.standard_scaler(bank, 'age')
27     bank = Standardize.standard_scaler(bank, 'campaign')
28     bank = Standardize.standard_scaler(bank, 'previous')
29     bank = Standardize.standard_scaler(bank, 'balance')
30     bank = Standardize.standard_scaler(bank, 'duration')
31     bank = Standardize.standard_scaler(bank, 'pdays')
32     bank = Standardize.standard_scaler(bank, 'day')
33
34     bank = ValueReplacer.tokenizer_by_column(bank, 'month')
35     bank = ValueReplacer.tokenizer_by_column(bank, 'contact')
```

Example 02

Chiller way

```
14 def another_pipeline():
15     bank = pd.read_csv('bank/bank-full.csv', delimiter=';')
16     need_tokenizer_columns = [
17         'housing',
18         'loan',
19         'default',
20         'marital',
21         'job',
22         'education'
23     ]
24
25     for col_name in need_tokenizer_columns:
26         bank = ValueReplacer.tokenizer_by_column(bank, col_name)
```

CLEAN CODE TIPS

- Clean code always looks like it was written by someone who cares.
- Make your code read like a story.
- The first rule of functions is that should be small.
 - A function should do one thing, and only one thing.
- Methods should be have verb or phrase names.
- Tips for Error Handling:
 - Avoid empty except blocks !!!!!
 - Avoid empty except blocks !!!!!
 - Avoid empty except blocks !!!!!



THANKS!

Do you have any questions?

CREDITS: This presentation template was created by [Slidesgo](#), including icons by [Flaticon](#), infographics & images by [Freepik](#)