Intro a Programación Orientada a Obxetos con Python:

Introducción a POO & Python

M. Torre Castro[1]

[1]Python Coruña

Charlas Python Coruña, 2023

Índice

Outline

Programación estructurada antes da POO

O pasado imperativo...

- Pra falar da Programación Orientada a Obxetos (POO ou OOP en inglés), antes imos introducir a causa da sua invención
- Os programas tradicionais antes da POO eran secuencias de definicións de datos e instruccións que operaban sobre eles
- As linguaxes non aportaban unha ferramenta que permita separar o código en unidades **conceptuais** (hai certas excepcións)

Programación estructurada antes da POO

O pasado imperativo...

- Pra falar da Programación Orientada a Obxetos (POO ou OOP en inglés), antes imos introducir a causa da sua invención
- Os programas tradicionais antes da POO eran secuencias de definicións de datos e instruccións que operaban sobre eles
- As linguaxes non aportaban unha ferramenta que permita separar o código en unidades **conceptuais** (hai certas excepcións)

Programación estructurada antes da POO

O pasado imperativo...

- Pra falar da Programación Orientada a Obxetos (POO ou OOP en inglés), antes imos introducir a causa da sua invención
- Os programas tradicionais antes da POO eran secuencias de definicións de datos e instruccións que operaban sobre eles
- As linguaxes non aportaban unha ferramenta que permita separar o código en unidades **conceptuais** (hai certas excepcións)

Prog. estructurada: Exemplo práctico

Imos ilustrar como se programaba antes da POO cun exemplo. Se desexa implementar un sistema dun servizo online onde hai un periodo de proba gratuito, pero só serán aptos pra optar a esta proba os usuarios maiores de idade que teñan rexistrada tarxeta de crédito.

Exemplo sen POO: Corpo principal

```
1   card_status = {True: 'válida', False: 'inválida'}
2
3   def main():
4       user_requests = []
5       while True:
6           try:
7               new_suscriptor = input_user_request()
8           except ValueError:
9               print('Formato erroneo. Usuario abortado')
10              continue
11          if new_suscriptor['username'] == '':
12              break
13          user_requests.append(new_suscriptor)
14          print('Usuario introducido')
15
16      for user in user_requests:
17          print(f"{user['username']} ten {age(user)} anos.")
18          print(f"Tarxeta {card_status[is_valid_card(user)]}.")
19          if not is_valid_card(user):
20              print('Tarxeta inválida. Usuario rexeitado pra periodo gratuito de proba')
21          elif age(user) < 18:
22              print('Edad insuficiente. Usuario rexeitado pra periodo gratuito de proba')
23          else:
24              print('Usuario aceptado pra periodo gratuito de proba')
```

Exemplo sen POO: Funcións auxiliares

```
1   def input_user_request():
2       username = input('\nIntroduzca nombre o deje en blanco para finalizar: ').strip()
3       if username == '':
4           return { 'username': ''}
5       string_date = input('Introduzca una fecha de nacimiento (yyyy/MM/dd): ').strip()
6       try:
7           input_date = dt.datetime(*[int(token) for token in string_date.split('/')])
8       except ValueError as e:
9           raise e
10      credit_card = input('Introduzca su tarjeta de credito (16 digitos): ').strip()
11      new_suscriptor = {
12          'username': username,
13          'birthdate': input_date,
14          'credit_card': credit_card
15      }
16      return new_suscriptor
17
18  def is_valid_card(user: dict):
19      return len(user['credit_card']) == 16
20
21  # Simplificación => NON USAR ISTO!
22  def age(userdict: dict) -> int:
23      interval = dt.datetime.now() - userdict['birthdate']
24      return interval.days // Person.DAYS_PER_YEAR
```

Puntos de mellora do código

- Dificultade na modelización, semántica e lexibilidade
- Dificultade na reutilización
- Dificultade na administración

Puntos de mellora do código

- Dificultade na modelización, semántica e lexibilidade
- Dificultade na reutilización
- Dificultade na administración

Puntos de mellora do código

- Dificultade na modelización, semántica e lexibilidade
- Dificultade na reutilización
- Dificultade na administración

Outline

Que é a Programación Orientada a Obxetos?

Cambiando o chip...

- Este enfoque permite agrupar o código en **obxetos**: un novo tipo de unidade de código
- Un obxeto agrupa datos e operacións
- O obxeto ven definido por unha **clase**, que define a estructura do obxeto e permite facer **instancias**

Que é a Programación Orientada a Obxetos?

Cambiando o chip...

- Este enfoque permite agrupar o código en **obxetos**: un novo tipo de unidade de código
- Un obxeto agrupa datos e operacións
- O obxeto ven definido por unha **clase**, que define a estructura do obxeto e permite facer **instancias**

Que é a Programación Orientada a Obxetos?

Cambiando o chip...

- Este enfoque permite agrupar o código en **obxetos**: un novo tipo de unidade de código
- Un obxeto agrupa datos e operacións
- O obxeto ven definido por unha **clase**, que define a estructura do obxeto e permite facer **instancias**

Obxeto e instancia

Figure:

Programar con clases

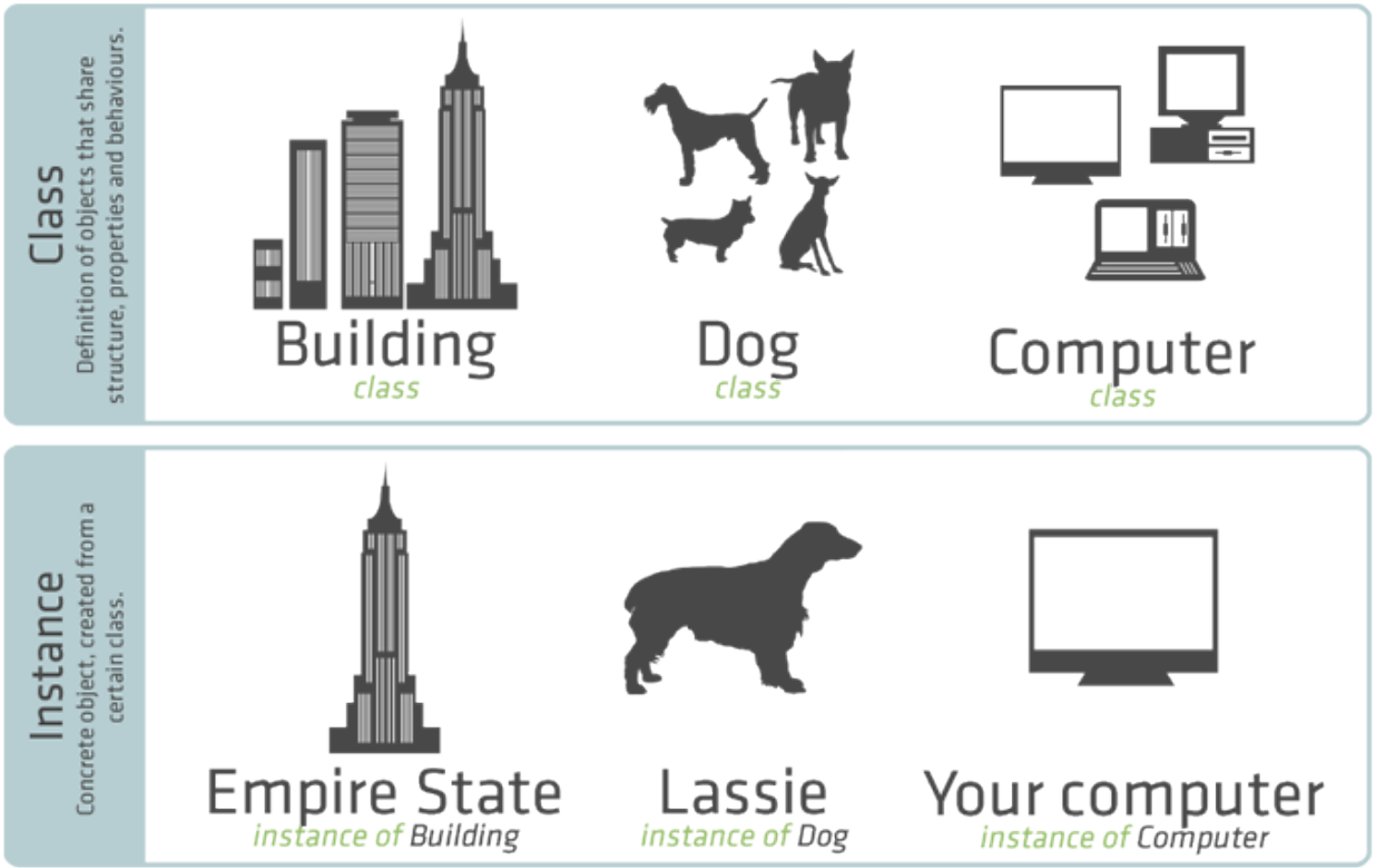

Figure:

Outline

Principios de POO

Figure:

Principios de POO

As características básicas da POO

- Encapsulación: datos e operacións relacionados van xuntos e son accedidos soamente pola clase
- Abstracción: os conceptos complexos quedan ocultos no obxeto ou clase que os abarca e que nos provee de atributos e métodos faciles de manexar
- Herencia: As clases poden compartir información (atributos) e comportamento (métodos) e facer xerarquias
- Polimorfismo: Os obxetos poden ter diferente comportamento según o contexto e tamén poden ser tratados igual ainda que os tipos sexan distintos

Principios de POO

As características básicas da POO

- Encapsulación: datos e operacións relacionados van xuntos e son accedidos soamente pola clase
- Abstracción: os conceptos complexos quedan ocultos no obxeto ou clase que os abarca e que nos provee de atributos e métodos faciles de manexar
- Herencia: As clases poden compartir información (atributos) e comportamento (métodos) e facer xerarquias
- Polimorfismo: Os obxetos poden ter diferente comportamento según o contexto e tamén poden ser tratados igual ainda que os tipos sexan distintos

Principios de POO

As características básicas da POO

- Encapsulación: datos e operacións relacionados van xuntos e son accedidos soamente pola clase
- Abstracción: os conceptos complexos quedan ocultos no obxeto ou clase que os abarca e que nos provee de atributos e métodos faciles de manexar
- Herencia: As clases poden compartir información (atributos) e comportamento (métodos) e facer xerarquias
- Polimorfismo: Os obxetos poden ter diferente comportamento según o contexto e tamén poden ser tratados igual ainda que os tipos sexan distintos

Principios de POO

As características básicas da POO

- Encapsulación: datos e operacións relacionados van xuntos e son accedidos soamente pola clase
- Abstracción: os conceptos complexos quedan ocultos no obxeto ou clase que os abarca e que nos provee de atributos e métodos faciles de manexar
- Herencia: As clases poden compartir información (atributos) e comportamento (métodos) e facer xerarquias
- Polimorfismo: Os obxetos poden ter diferente comportamento según o contexto e tamén poden ser tratados igual ainda que os tipos sexan distintos

Exemplo de obxeto

O único caso onde unha persoa debe ser tratada coma un obxeto

```
1   class Person():
2       DAYS_PER_YEAR = 365
3       def __init__(self, name, birth_dt):
4           self.name = name
5           self.birth_dt = birth_dt
6
7       def age(self):
8           interval = dt.datetime.now() - self.birth_dt
9           return interval.days // Person.DAYS_PER_YEAR
10
11  p = Person('Alex', dt.datetime(2003, 5, 13)) # Instancia creada!
12  print(f'{p.name} ten {p.age()} anos ')   # Alex ten 20 anos
```

Principios de POO: Herencia

O pai que non sae ao fillo é un porco

- A herencia é unha cualidade da POO pola cal unha clase pode herdar atributos e funcións doutra
- As clases conforman así xerarquías onde a filla recibe da nai
- Así, se precisamos ampliar ou personalizar máis o seu funcionamento podemos engadir eses detalles en clases descendentes
- A clase descendente pode redefinir as funcións ou amplialas

Principios de POO: Herencia

O pai que non sae ao fillo é un porco

- A herencia é unha cualidade da POO pola cal unha clase pode herdar atributos e funcións doutra
- As clases conforman así xerarquías onde a filla recibe da nai
- Así, se precisamos ampliar ou personalizar máis o seu funcionamento podemos engadir eses detalles en clases descendentes
- A clase descendente pode redefinir as funcións ou amplialas

Principios de POO: Herencia

O pai que non sae ao fillo é un porco

- A herencia é unha cualidade da POO pola cal unha clase pode herdar atributos e funcións doutra
- As clases conforman así xerarquías onde a filla recibe da nai
- Así, se precisamos ampliar ou personalizar máis o seu funcionamento podemos engadir eses detalles en clases descendentes
- A clase descendente pode redefinir as funcións ou amplialas

Principios de POO: Herencia

O pai que non sae ao fillo é un porco

- A herencia é unha cualidade da POO pola cal unha clase pode herdar atributos e funcións doutra
- As clases conforman así xerarquías onde a filla recibe da nai
- Así, se precisamos ampliar ou personalizar máis o seu funcionamento podemos engadir eses detalles en clases descendentes
- A clase descendente pode redefinir as funcións ou amplialas

Herencia: Exemplo ilustrativo

Eramos poucos e pariu a avoa

Agora pasamos a ilustrar unha xerarquía de herencia cunha idea semellante á anterior. Unha plataforma que ofrece diversos servizos online.

Se precisa diferenciar os tipos de usuarios pra proveerlles distintos servizos con distintas características.

Xerarquía

Figure:

Clase raíz/nai/pai

```
1   class BronzeUser(User):
2       price = 10
3       num_disp = 1
4       music_limit = 2
5       video_limit = 2
6       media_limit = 2
7
8       def __init__(self, username, birthdate, credit_card):
9           super().__init__(username, birthdate, credit_card)
10
11      def pay(self):
12          print(f'Cobrando cuota mensual de {self.username}: {self.price}')
13          print(f'Tarxeta: {self.credit_card}')
14          self.show_num_gadgets()
15
16      def show_num_gadgets(self):
17          print(f'Num dispositivos: {self.num_disp}. ')
18
19      def stream_music(self):
20          print(f'Escoitando musica. Capacidade ata {self.music_limit} GBs')
21
22      def store_media(self):
23          print(f'Subindo fotos ata un máximo de {self.media_limit} GBs')
```

Clase filla (2ª xeración)

```
1   class SilverUser(BronzeUser):
2       price = 20
3       num_disp = 2
4       music_limit = 3
5       video_limit = 3
6       media_limit = 3
7
8       def store_media(self):
9           super().store_media()
10          print(f'¡Lembra que como {self.__class__.__name__} tamén podes subir videos!')
11
12      def stream_video(self):
13          print(f'Reproducindo videos con acceso ao catálogo básico.'
14                f'Descarga ata {self.video_limit} GBs')
```

Clase ¿nieta? (3ª xeración)

```
1   class GoldUser(SilverUser):
2       price = 50
3       num_disp = 4
4       music_limit = 10
5       video_limit = 10
6       media_limit = 10
7
8       def stream_music(self):
9           super().stream_music()
10          print('Podes solicitar que U2 vaia a tocar á tua casa')
11
12      def store_media(self):
13          super().store_media()
14          print('Subindo fotos ata un máximo de 5 GBs')
15
16      def stream_video(self):
17          super().stream_video()
18          print('Lembra que como GoldUser tamén tes acceso aos últimos estrenos')
19
20      def stream_gaming(self):
21          print('Xogando sen límite de catálogo')
```

¡Coidado coa herencia!

- A herencia é unha ferramenta moi util e poderosa, pero non debemos abusar dela cada vez que duas clases teñan algo en común
- Pode conducir a deseños moi rixidos
- Como regla intuitiva, tratar de usar herencia só se vemos que na realidad a clase filla reune realmente todo o da clase nai e cumpre unha función análoga
- Eu aquí sempre me lembro das palabras de alguén moi sabio...

¡Coidado coa herencia!

- A herencia é unha ferramenta moi util e poderosa, pero non debemos abusar dela cada vez que duas clases teñan algo en común
- Pode conducir a deseños moi rixidos
- Como regla intuitiva, tratar de usar herencia só se vemos que na realidad a clase filla reune realmente todo o da clase nai e cumpre unha función análoga
- Eu aquí sempre me lembro das palabras de alguén moi sabio...

¡Coidado coa herencia!

- A herencia é unha ferramenta moi util e poderosa, pero non debemos abusar dela cada vez que duas clases teñan algo en común
- Pode conducir a deseños moi rixidos
- Como regla intuitiva, tratar de usar herencia só se vemos que na realidad a clase filla reune realmente todo o da clase nai e cumpre unha función análoga
- Eu aquí sempre me lembro das palabras de alguén moi sabio...

¡Coidado coa herencia!

- A herencia é unha ferramenta moi util e poderosa, pero non debemos abusar dela cada vez que duas clases teñan algo en común
- Pode conducir a deseños moi rixidos
- Como regla intuitiva, tratar de usar herencia só se vemos que na realidad a clase filla reune realmente todo o da clase nai e cumpre unha función análoga
- Eu aquí sempre me lembro das palabras de alguén moi sabio...

A herencia non sempre compensa

"¡Tanto super y tanta hostia!"

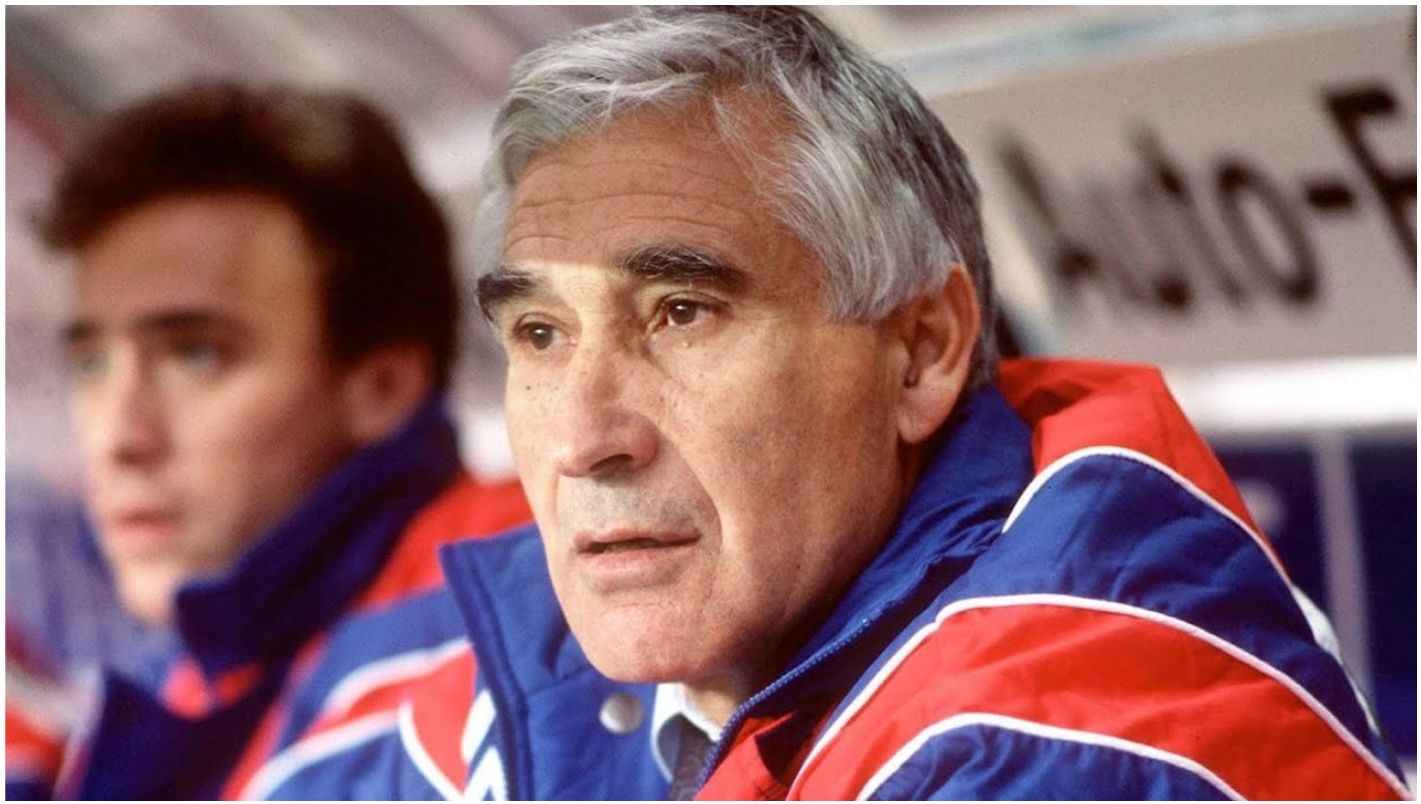

Figure:

Outline

Exemplo sen POO: Corpo principal

```
1   card_status = {True: 'válida', False: 'inválida'}
2
3   def main():
4       user_requests = []
5       while True:
6           try:
7               new_suscriptor = input_user_request()
8           except ValueError:
9               print('Formato erroneo. Usuario abortado')
10              continue
11          if new_suscriptor['username'] == '':
12              break
13          user_requests.append(new_suscriptor)
14          print('Usuario introducido')
15      for user in user_requests:
16          print(f"{user['username']} ten {age(user)} anos.")
17          print(f"Tarxeta  {card_status[is_valid_card(user)]}.")
18          if not is_valid_card(user):
19              print('Tarxeta inválida. Usuario rexeitado pra periodo gratuito de proba')
20          elif age(user) < 18:
21              print('Edad insuficiente. Usuario rexeitado pra periodo gratuito de proba')
22          else:
23              print('Usuario aceptado pra periodo gratuito de proba')
```

Exemplo sen POO: Funcións auxiliares

```
1   def input_user_request():
2       username = input('\nIntroduzca nombre o deje en blanco para finalizar: ').strip()
3       if username == '':
4           return { 'username': ''}
5       string_date = input('Introduzca una fecha de nacimiento (yyyy/MM/dd): ').strip()
6       try:
7           input_date = dt.datetime(*[int(token) for token in string_date.split('/')])
8       except ValueError as e:
9           raise e
10      credit_card = input('Introduzca su tarjeta de credito (16 digitos): ').strip()
11      new_suscriptor = {
12          'username': username,
13          'birthdate': input_date,
14          'credit_card': credit_card
15      }
16      return new_suscriptor
17
18  def is_valid_card(user: dict):
19      return len(user['credit_card']) == 16
20
21  # Simplificación => NON USAR ISTO!
22  def age(userdict: dict) -> int:
23      interval = dt.datetime.now() - userdict['birthdate']
24      return interval.days // Person.DAYS_PER_YEAR
```

Exemplo con POO: Clase User

```
1   class User:
2       def __init__(self, username, birthdate, credit_card):
3           self.username = username
4           self.birthdate = birthdate
5           self.credit_card = credit_card
6
7       def is_valid_card(self):
8           return len(self.credit_card) == 16
9
10      def is_valid_card_as_string(self):
11          if self.is_valid_card():
12              return 'válida'
13          else:
14              return 'inválida'
15
16      def is_adult(self):
17          return self.age() >= 18
18
19      def age(self) -> int:
20          interval = dt.datetime.now() - self.birthdate
21          return interval.days // Person.DAYS_PER_YEAR
```

Exemplo con POO: Corpo principal

```
1  def main():
2      user_requests = []
3      while True:
4          try:
5              new_user = input_user_request()
6          except ValueError:
7              print('Formato erroneo. Usuario abortado')
8              continue
9          if not new_user:
10             break
11     user_requests.append(new_user)
12     print('Usuario introducido')
13
14     for user in user_requests:
15         print(f"{user.username} ten {user.age()} anos")
16         print(f"Tarxeta  {user.is_valid_card_as_string()}.")
17         if not user.is_valid_card():
18             print('Tarxeta inválida. Usuario rexeitado pra periodo gratuito de proba')
19         elif not user.is_adult():
20             print('Edad insuficiente. Usuario rexeitado pra periodo gratuito de proba')
21         else:
22             print('Usuario aceptado pra periodo gratuito de proba')
```

Mellora palpable

Figure:

Resumo

- Isto foi unha intro pero hai moitos detalles que aprender na POO
- Desde boas prácticas e principios (composición sobre herencia, as clases deben estar abertas pra ser ampliadas pero pechadas pra ser modificadas)
- Ata patróns de deseño (buscade a miña charla do ano pasado no repo =:-D)

Resumo

- Isto foi unha intro pero hai moitos detalles que aprender na POO
- Desde boas prácticas e principios (composición sobre herencia, as clases deben estar abertas pra ser ampliadas pero pechadas pra ser modificadas)
- Ata patróns de deseño (buscade a miña charla do ano pasado no repo =:-D)

Resumo

- Isto foi unha intro pero hai moitos detalles que aprender na POO
- Desde boas prácticas e principios (composición sobre herencia, as clases deben estar abertas pra ser ampliadas pero pechadas pra ser modificadas)
- Ata patróns de deseño (buscade a miña charla do ano pasado no repo =:-D)

Ruegos e preguntas

Figure:

Pra ler máis I


Fluent Python (O'Reilly) - by Ramalho


Learning Python (O'Reilly) - by Mark Lutz