

INSTITUTO FEDERAL CATARINENSE - CAMPUS BLUMENAU
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PÂMELA CRISTINA SPERAFICO

Reflexão Computacional e Python: mini-relato

RESUMO

A reflexão computacional diz respeito à capacidade que o programa tem de raciocinar sobre si e afetar a si próprio. Dentro desse conceito de reflexão, existem várias características e definições que a permeiam como metadados, modificação de código em tempo de execução, entre outros. No presente trabalho, estes conceitos serão apresentados de modo sucinto, sendo que, serão incluídos também exemplos práticos que foram construídos para embasar a respeito das características e definições que permeiam a reflexão.

SUMÁRIO

1 REFLEXÃO	4
1.1 REFLEXÃO E INTROSPECÇÃO	4
2 A REFLEXÃO DENTRO DO PYTHON	6
2.1 METACLASSES	6
2.2 METADADOS	7
2.3 EXEMPLOS PRÁTICOS	8
REFERÊNCIAS	11
UNDERSTANDING OBJECT INSTANTIATION AND METACLASSES IN PYTHON - HTTPS://WWW.HONEYBADGER.IO/BLOG/PYTHON-INSTANTIATION-METAClass/	11

1 REFLEXÃO

Reflexão computacional é a capacidade que o programa tem de modificar-se diretamente durante a sua execução e de outros, ou seja, o sistema computacional raciocina sobre si mesmo e afeta a si próprio. Linguagens que possuem essas características são intituladas metalinguagens e a linguagem do programa que foi manipulado é chamada de linguagem objeto.

Com isso, a reflexão computacional é de grande interesse para a implementação de sistemas; esse interesse é prático devido a sua capacidade para representar o sistema, suas funcionalidades e sua implementação, dentro do próprio sistema. Dessa forma, as linguagens orientadas a objetos podem ter a reflexão facilmente implementada devido a sua natureza, na qual dados e procedimentos são distribuídos nos objetos. Assim, os programas que possuem a capacidade de modificarem seu próprio código fonte (ou de outro programa) durante o tempo de execução são chamados de metaprogramas.

A reflexão computacional tem suas classes, métodos, objetos e atributos retratados por meio de metaclasses e metaobjetos. Com isso, podemos afirmar que um sistema reflexivo possui três estágios para alcançar a estabilidade; o primeiro seria atingir uma descrição abstrata do sistema de modo com que ela seja efetiva, permitindo assim operações sobre ela. Por conseguinte, utiliza-se essa efetividade para realizar certa manipulação e por fim, essa manipulação é retornada com seus resultados da reflexão computacional.

Dado o exposto, pode-se afirmar que a reflexão permite, em tempo de execução, a verificação da classe de um objeto, sendo que este objeto pode ser construído também em tempo de execução. Métodos de determinadas classes também podem ser examinados em tempo de execução e qualquer método de um objeto pode ser invocado, também em tempo de execução.

As vantagens a respeito do uso da reflexão são sua flexibilidade para a generalização e sua alta capacidade de manipulação. A reflexão é uma ferramenta que provém a construção de códigos reutilizáveis, permitindo que sejam aplicados nos mais diversos conceitos.

1.1 REFLEXÃO E INTROSPECÇÃO

Dentro da reflexão, devemos abrir espaço também para a discussão da introspecção e sua comparação com a reflexão. A introspecção é a habilidade na qual um programa examina

suas próprias estruturas de dados e também sua programação em tempo de execução; assim, dentro do Python, que será a linguagem para a implementação dos exemplos, possui como método mais comum de introspecção a função `dir`. Assim, esse método tenta retornar uma lista de atributos válidos do objeto. Dessa forma, se o objeto não tiver o método `__dir__()`, este método tenta encontrar informações do atributo `__dict__` (se definido) e do tipo de objeto. Nesse caso, a lista retornada de `dir()` pode não estar completa.

Assim, a introspecção possui apenas um diferencial no que diz respeito à reflexão, e esse diferencial seria a capacidade de modificação das estruturas de dados e de programação durante a execução.

2 A REFLEXÃO DENTRO DO PYTHON

O Python é uma linguagem de programação orientada a objetos que possui como características ser ágil, objetiva e fácil, sendo que a simplicidade é seu ponto forte. Dessa forma, algumas das características e aspectos dessa linguagem serão apresentados para a compreensão do recurso reflexivo presente nessa linguagem.

2.1 METACLASSES

Metaclasses são definidas como a classe de uma classe. Dessa maneira, uma metaclasses define o comportamento de um programa. Existem algumas diferentes metaclasses dentro do Python, contudo, as principais são o `object` e o `type`. Sendo assim, o `object` diz respeito a todo objeto que é criado baseado em algum objeto que terá sua metaclasses definida pelo menos nome; já o `type` diz respeito aos tipos de dados que são criados a partir de um objeto e é definido pela metaclasses de mesmo nome. Assim, pode-se afirmar que todo elemento criado em Python herdará um objeto `type` que por conseguinte herdará de `object`, sendo esse o topo da hierarquia das metaclasses.

Assim, metaclasses são indispensáveis caso deseje ter objetos de classes que possuem um comportamento “personalizado”, uma vez que, o comportamento de um objeto depende de métodos do tipo de objeto, e esse tipo de objeto de classes é sinônimo de metaclasses.

Agora, para exemplificar, estarei apresentado abaixo alguns dos métodos definidos pelas metaclasses, que ditam o comportamento do sistema.

- `__new__()`: é chamado na criação da instância e passa qualquer argumento na criação para o inicializador;
- `__init__()`: é o inicializador da classe, ele é transmitido com o que quer que o construtor primário tenha sido chamado;
- `__del__()`: é o que destrói e deleta, contudo, sua utilização deve ser feita com cautela;
- `__str__()`: é o comportamento para quando uma string é chamada em uma instância da classe;
- `__eq__()`: é o que define o comportamento para o operador de igualdade, `==` ;
- `__call__()`: é o que permite que uma instância de uma classe seja chamada como uma função.

2.2 METADADOS

Metadados são dados que refletem algum aspecto do sistema em si e não necessariamente do domínio que ele está inserido. Dentro do Python, metadados do sistema podem ser utilizados e acessados a partir dos objetos criados nas metaclasses.

Dessa forma, a seguir pode-se encontrar alguns dos metadados mais comuns. Abaixo, estão alguns exemplos.

- `__name__`: obtém seu valor dependendo de como executamos o script que o contém;
- `__bases__`: são as superclasses da classe atual;
- `__class__`: é a classe na qual o objeto realiza sua instância;
- `__dict__`: é um dicionário ou outro objeto de mapeamento usado para armazenar os atributos de um objeto;
- `__defaults__`: são as padronizações definidas dentro dos métodos;

Assim, agora com o conceito de metadados e metaclasses, podemos refletir acerca da alteração de código em tempo de execução. Essa modificação ocorre devido a linguagem possuir metaclasses, permitindo que os objetos sofram alterações. Para que isso ocorra, basta que ocorra a alteração dos metadados e também dos métodos que são herdados das metaclasses.

Para finalizar, existem alguns metadados que podem ser acessados por meio de funções built-in, que são funções que permitem somente a leitura dos dados e não realizam a modificação deles. Abaixo, estão alguns exemplos.

- `type()`: retorna o tipo de classe do objeto que foi passado como parâmetro.
- `id()`: retorna o inteiro único de um objeto;
- `isinstance()`: retorna um valor lógico de acordo com o tipo do objeto que foi especificado;
- `issubclass()`: verifica se o argumento da classe é uma subclasse;
- `object()`: é a base para todas as classes, e contém propriedades e métodos padrões para todas as classes, contudo, não se pode adicionar novas propriedades ou métodos ao `object()`.

2.3 EXEMPLOS PRÁTICOS

Essa seção é dedicada a colocar em prática alguns dos conceitos da reflexão que foram expostos anteriormente. Assim, constarão aqui somente alguns prints para exemplificar, contudo, os códigos podem ser acessados na íntegra por meio do seguinte link do GitHub: <https://github.com/pamelasperafico/Program-Orientada-a-Objetos-II/tree/main/AV1%20-%20C%C3%B3digo%20reflex%C3%A3o%20e%20minirelato>.

Para começar, trago um exemplo de `__new__()` que é contido dentro da Metaclass em Python. Dentro do método `__new__` da classe `Lagomorfos`, chama-se o método `__new__` da classe de objeto usando `super().__new__(classe)`. Assim, o método `__new__` da classe de objeto cria e retorna a instância da classe, que é passada como um argumento para o método `__new__`. Dessa maneira, conforme o programa vai passando classe, que é a referência da classe `Lagomorfos`, o método `__new__` do objeto retornará uma instância do tipo `Lagomorfos`. Assim, para realizar a alocação da memória chama-se o método `__new__` da classe de objeto dentro do método `__new__` sobrescrito. O método `__new__` da classe `Lagomorfos` modifica o objeto retornado do método `__new__` da classe de objeto e adiciona a propriedade `nome` a ele. Assim, todos os objetos criados com a classe `Lagomorfos` terão uma propriedade `nome`. Veja abaixo um recorte do código e de sua compilação.



```
lagomorfos.py X
C: > Users > spera > Desktop > lagomorfos.py > ...
1 class Lagomorfos:
2     def __new__(classe, classifica_lagomorf=None):
3         objeto = super().__new__(classe)
4         if classifica_lagomorf:
5             objeto.nome = classifica_lagomorf
6         else:
7             objeto.nome = "Coelho"
8
9         print(type(objeto))
10        return objeto
11
12
13 coelho = Lagomorfos()
14
15 print(coelho.nome)
16
17 lebre = Lagomorfos("lebre")
18 print(lebre.nome)
```

Figura 1 - Exemplo Lagomorfos


```
<class '__main__.Lagomorfos'>
Coelho
<class '__main__.Lagomorfos'>
lebre
```

Figura 2 - Compilação Lagomorfos

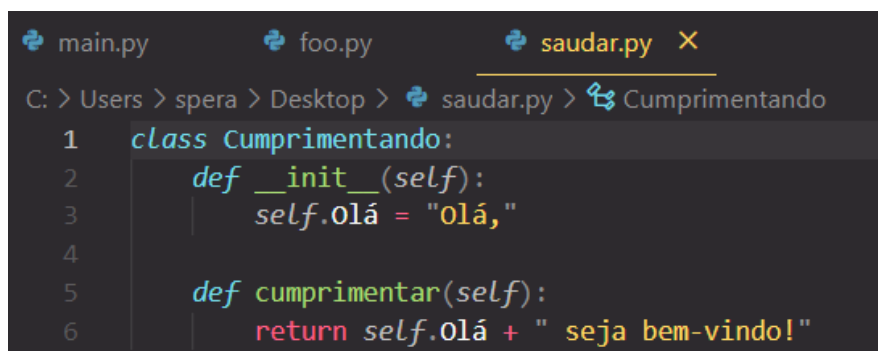
Com isso, outro exemplo que trago é um exemplo básico da reflexão, que é composto pelas classes main, foo e saudar dentro do repositório do GitHub. Esse exemplo reforça um conceitos básicos utilizando a impressão de uma saudação em tela: “Olá, seja bem-vindo!”. Ao observar o main, percebe-se o acesso e exibição de metadados que constam nas outras classes, e esse são exibidos a partir da função print(). Esse exemplo reforça a questão da recuperação de dados vindos de outras classes.

```
main.py x foo.py saudar.py
C: > Users > spera > Desktop > main.py > ...
1  import sys
2
3  from foo import Foo
4  from saudar import Cumprimentando
5
6  # Após compilar, uma saudação é mostrada em tela
7
8  if __name__ == "__main__":
9      classe = sys.argv[0].capitalize()
10     classe = globals()[classe if classe in ["Foo"] else "Cumprimentando"]
11     instancia = classe()
12     print(instancia.cumprimentar())
```

Figura 3 - Exemplo main

```
main.py foo.py x saudar.py
C: > Users > spera > Desktop > foo.py > ...
1  from saudar import Cumprimentando
2
3
4  class Foo(Cumprimentando):
5      def cumprimentar(self):
6          return self.ola + "Foo"
```

Figura 4 - Exemplo foo

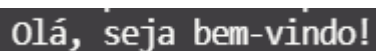


```

1 class Cumprimentando:
2     def __init__(self):
3         self.Olá = "Olá,"
4
5     def cumprimentar(self):
6         return self.Olá + " seja bem-vindo!"

```

Figura 5 - Exemplo saudar



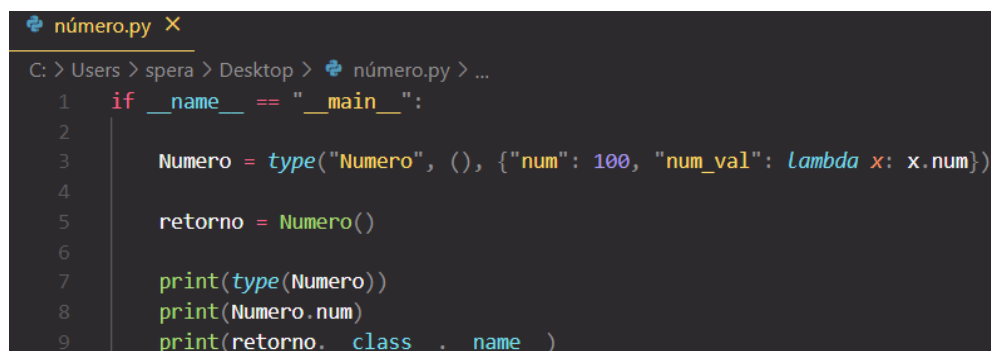
```

Olá, seja bem-vindo!

```

Figura 6 - Resultado da compilação main

Por fim, o último exemplo é a classe número, que também consta no repositório do GitHub. Esse exemplo final demonstra como se faz a relação entre classes e metaclasses, que pode ser visualizado a partir dos resultados da execução.

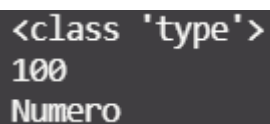


```

1 if __name__ == "__main__":
2
3     Numero = type("Numero", (), {"num": 100, "num_val": lambda x: x.num})
4
5     retorno = Numero()
6
7     print(type(Numero))
8     print(Numero.num)
9     print(retorno.__class__.__name__)

```

Figura 7 - Exemplo número



```

<class 'type'>
100
Numero

```

Figura 8 - Resultado da compilação

REFERÊNCIAS

Computational Reflection in Class based Object Oriented Languages - https://www.researchgate.net/publication/234811008_Computational_reflection_in_class_based_object-oriented_languages

Um estudo sobre Metaprogramação: as linguagens de programação Python e Ruby - <http://conic-semesp.org.br/anais/files/2014/trabalho-1000018336.pdf>

Utilização de reflexão computacional em um gerenciador de arquivos distribuídos - <http://fbarth.net.br/docs/lop2000.pdf>

Reflection in Python - <https://www.geeksforgeeks.org/reflection-in-python/>

Python Metaclasses - <https://realpython.com/python-metaclasses/>

Understanding Object Instantiation and Metaclasses in Python - <https://www.honeybadger.io/blog/python-instantiation-metaclass/>

Data Model - <https://docs.python.org/3/reference/datamodel.html>