Aula 09

Revisão

Mais alguns exemplos de uso de herança:

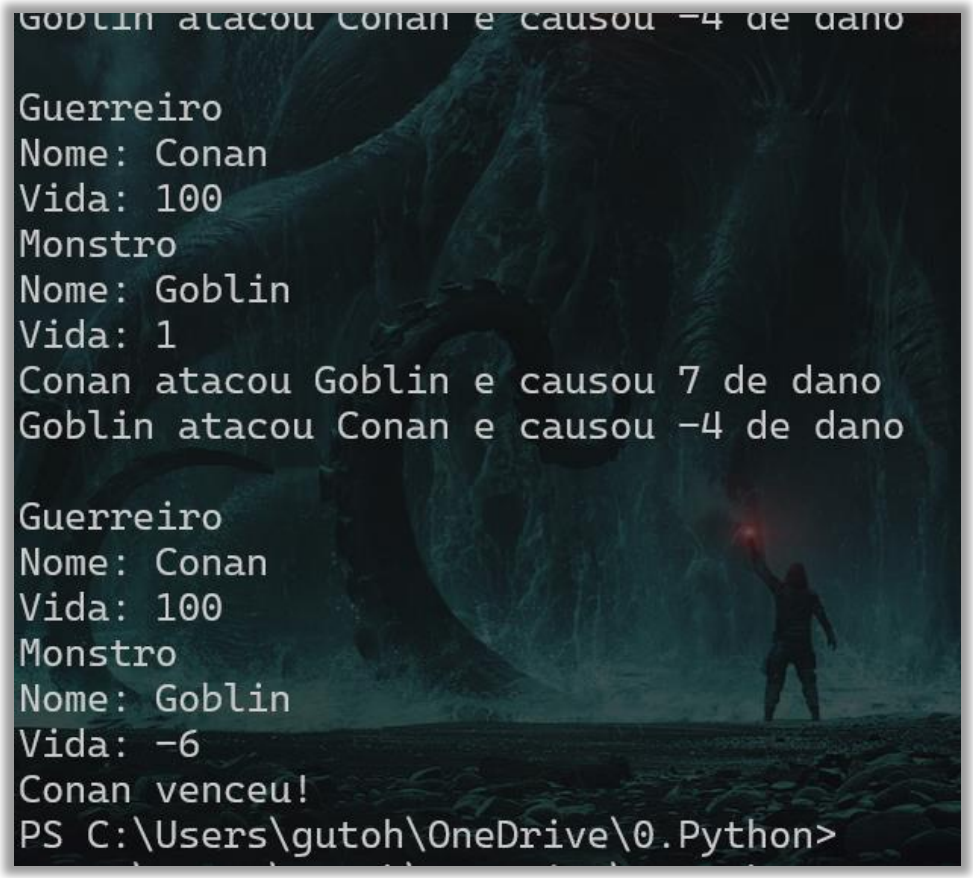
```python
from random import randint


class Personagem:
    """classe Personagem"""

    def __init__(self, p_nome: str, p_vida: int) -> None:
        self.nome: str = p_nome
        self.vida: int = p_vida
        self.ataque: int = 0
        self.defesa: int = 0

    def ataca(self, p_oponente: 'Personagem') -> None:
        """método ataca"""
        dano: int = self.ataque - p_oponente.defesa
        print(f'{self.nome} atacou {p_oponente.nome} e causou {dano} de dano')
        if dano > 0:
            p_oponente.vida -= dano

    def esta_vivo(self) -> bool:
        """método esta_vivo"""
        return self.vida > 0

    def __str__(self) -> str:
        return f'Nome: {self.nome}\nVida: {self.vida}'
```

```python
class Guerreiro(Personagem):
    """classe Guerreiro"""

    def __init__(self, p_nome: str, p_vida: int) -> None:
        super().__init__(p_nome, p_vida)
        self.ataque: int = 10
        self.defesa: int = 5

    def __str__(self) -> str:
        return f'Guerreiro\n{super().__str__()}'
```

```python
class Monstro(Personagem):
    """classe Monstro"""

    def __init__(self, p_nome: str, p_vida: int) -> None:
        super().__init__(p_nome, p_vida)
        self.ataque: int = randint(1, 4)
        self.defesa: int = randint(1, 4)

    def __str__(self) -> str:
        return f'Monstro\n{super().__str__()}'
```

```python
from classes import Guerreiro, Monstro

# Instanciando objetos
conan: Guerreiro = Guerreiro('Conan', 100)
goblin: Monstro = Monstro('Goblin', 50)

while True:
    print(conan)
    print(goblin)
    if not conan.esta_vivo():
        print('Goblin venceu!')
        break
    if not goblin.esta_vivo():
        print('Conan venceu!')
        break
    conan.ataca(goblin)
    goblin.ataca(conan)
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python main.py
Guerreiro
Nome: Conan
Vida: 100
Monstro
Nome: Goblin
Vida: 50
Conan atacou Goblin e causou 7 de dano
Goblin atacou Conan e causou -4 de dano

Guerreiro
Nome: Conan
Vida: 100
Monstro
Nome: Goblin
Vida: 43
Conan atacou Goblin e causou 7 de dano
Goblin atacou Conan e causou -4 de dano

Guerreiro
```



Módulo datetime

O módulo datetime é uma biblioteca integrada do Python que fornece classes para trabalhar com datas, horários e operações relacionadas. Ele permite a criação, manipulação, formatação e cálculo de datas e horários de forma eficiente.

A classe principal do módulo é chamada datetime, que combina informações de data e hora em um único objeto. Essa classe contém atributos como ano, mês, dia, hora, minuto, segundo e microssegundo. Você pode criar um objeto datetime especificando esses valores, ou usar os métodos fornecidos para obter a data e hora atuais.

Aqui está um exemplo de como criar um objeto datetime com uma data específica:

```
from datetime import datetime

data = datetime(2023, 5, 18)
print(data)
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python data.py
2023-05-18 00:00:00
PS C:\Users\gutoh\OneDrive\0.Python>
```

Além da classe datetime, o módulo datetime também fornece outras classes úteis, como date (para manipulação de datas), time (para manipulação de horários) e timedelta (para representar diferenças entre datas ou horários).

O módulo datetime também possui vários métodos para manipulação de datas e horários. Aqui estão alguns exemplos:

```python
from datetime import datetime, timedelta

# Obter a data e hora atuais
agora: datetime = datetime.now()
print(agora)

# Obter apenas a data
data_atual: date = agora.date()
print(data_atual)

# Obter apenas a hora
hora_atual: time = agora.time()
print(hora_atual)

# Adicionar um intervalo de tempo a uma data
data_futura: datetime = agora + timedelta(days=7)
print(data_futura)
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python data.py
2023-05-18 16:55:07.678848
2023-05-18
16:55:07.678848
2023-05-25 16:55:07.678848
PS C:\Users\gutoh\OneDrive\0.Python>
```

O módulo datetime também permite formatar datas e horários de várias maneiras usando a função strftime() e fazer o parsing de strings para objetos datetime usando a função strptime().

Strings e datetime

Para converter uma string para um objeto datetime no Python, você pode usar o método strptime() do módulo datetime. O strptime() permite fazer o parsing (interpretação) de uma string formatada em uma data e hora.

Aqui está um exemplo de como converter uma string em um objeto datetime:

```python
from datetime import datetime

data_string = "18/05/2023 19:30:00"
data: datetime = datetime.strptime(data_string, "%d/%m/%Y %H:%M:%S")
print(data)
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python data.py
2023-05-18 19:30:00
PS C:\Users\gutoh\OneDrive\0.Python>
```

Nesse exemplo, a string "18/05/2023 19:30:00" é convertida em um objeto datetime com a mesma data e hora. O segundo argumento do strptime() é o formato da string que você está passando. No exemplo, "%d/%m/%Y %H:%M:%S" é o formato que corresponde à string "18/05/2023 19:30:00". Os códigos de formatação utilizados têm os seguintes significados:

- %Y: ano com quatro dígitos.
- %m: mês com dois dígitos.
- %d: dia com dois dígitos.
- %H: hora em formato de 24 horas.
- %M: minuto com dois dígitos.
- %S: segundo com dois dígitos.

Certifique-se de que o formato especificado corresponda exatamente ao formato da string que você está convertendo. Caso contrário, ocorrerá um erro.

Depois de converter a string em um objeto datetime, você pode manipulá-lo conforme necessário, usar seus atributos ou realizar operações com outros objetos datetime.

É importante notar que o método strptime() só é usado para converter uma string em um objeto datetime. Se você deseja formatar um objeto datetime em uma string, você pode usar o método strftime().

Para converter um objeto datetime em uma string formatada no Python, você pode usar o método strftime() do módulo datetime. O strftime() permite formatar um objeto datetime de acordo com um formato especificado e retornar uma string representando a data e hora nesse formato.

Aqui está um exemplo de como converter um objeto datetime em uma string:

```python
from datetime import datetime

data = datetime(2023, 5, 18, 19, 30, 0)
print(data)
data_string: str = data.strftime("%d/%m/%Y %H:%M:%S")
print(data_string)
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python data.py
2023-05-18 19:30:00
18/05/2023 19:30:00
PS C:\Users\gutoh\OneDrive\0.Python>
```

Nesse exemplo, o objeto datetime data é convertido em uma string formatada no formato "%d/%m/%Y %H:%M:%S". A string resultante será "18/05/2023 19:30:00".

Você pode personalizar o formato da string conforme suas necessidades. Existem muitos outros códigos de formatação disponíveis para representar diferentes partes da data e hora. Certifique-se de utilizar os códigos corretos para obter o resultado desejado.

Mais Métodos Mágicos

Abaixo temos mais alguns métodos mágicos do Python. Com eles, podemos personalizar o comportamento dos nossos objetos com cada um dos operadores matemáticos.

- __add__(self, outro): Este método é invocado quando o operador de adição (+) é usado para somar dois objetos. Ele permite que você defina o comportamento da adição para uma classe personalizada.

- __sub__(self, outro): Semelhante ao método __add__, o método __sub__ é chamado quando o operador de subtração (-) é usado para subtrair dois objetos.

- __mul__(self, outro): O método __mul__ é acionado quando o operador de multiplicação (*) é usado para multiplicar dois objetos.

- __truediv__(self, outro): Este método é chamado quando o operador de divisão (/) é usado para dividir dois objetos. Ele retorna o resultado como um número de ponto flutuante (float).

- __floordiv__(self, outro): O método __floordiv__ é usado quando o operador de divisão de chão (//) é usado para realizar uma divisão inteira entre dois objetos. Ele retorna o resultado como um número inteiro, truncando o valor decimal.

- __mod__(self, outro): O método __mod__ é invocado quando o operador de módulo (%) é usado para calcular o resto da divisão entre dois objetos.

- __len__(self): O método __len__ retorna o tamanho de um objeto quando a função embutida len() é chamada nele. Ele permite que você personalize a lógica de determinação do tamanho de um objeto.

Ao implementar esses métodos mágicos em uma classe personalizada, você pode controlar o comportamento desses operadores para os objetos dessa classe, permitindo operações aritméticas e outras manipulações de acordo com a semântica desejada.

Como eles são muito parecidos, vamos ver os métodos add e len:
__add__

```python
class Ponto:
    def __init__(self, x: int, y: int) -> None:
        self.x: int = x
        self.y: int = y

    def __add__(self, outro) -> "Ponto":
        if isinstance(outro, Ponto):
            novo_x: int = self.x + outro.x
            novo_y: int = self.y + outro.y
            return Ponto(novo_x, novo_y)
        raise TypeError("Apenas objetos da classe Ponto podem ser somados.")

# Criando dois objetos da classe Ponto
ponto1 = Ponto(2, 3)
ponto2 = Ponto(5, 7)

# Usando o operador de adição para somar os pontos
resultado: Ponto = ponto1 + ponto2

# Exibindo o resultado
print(resultado.x, resultado.y)  # Saída: 7 10
```

Nesse exemplo, temos a classe Ponto que representa um ponto no plano cartesiano com coordenadas x e y. O método __add__() é implementado para permitir a adição de dois objetos da classe Ponto. No método, verificamos se outro é uma instância da classe Ponto. Se for, somamos as coordenadas x e y dos dois pontos e criamos um novo objeto Ponto com as coordenadas resultantes. Caso outro não seja um objeto da classe Ponto, lançamos uma exceção de tipo.

Depois, criamos dois objetos Ponto e usamos o operador de adição (+) para somá-los, o que aciona o método __add__(). O resultado é armazenado na variável resultado e, em seguida, exibimos as coordenadas x e y do ponto resultante.

__len__

```python
class ListaPersonalizada:
    def __init__(self, elementos:list[int]) -> None:
        self.elementos: list[int] = elementos

    def __len__(self) -> int:
        return len(self.elementos)

# Criando um objeto da classe ListaPersonalizada
minha_lista = ListaPersonalizada([1, 2, 3, 4, 5])

# Usando a função embutida len() para obter o tamanho da lista personalizada
tamanho: int = len(minha_lista)

# Exibindo o tamanho da lista personalizada
print(tamanho)  # Saída: 5
```

Nesse exemplo, temos a classe ListaPersonalizada que representa uma lista personalizada. No método __len__() dessa classe, simplesmente retornamos o tamanho da lista elementos utilizando a função embutida len().

Em seguida, criamos um objeto minha_lista da classe ListaPersonalizada, passando uma lista de elementos como argumento para o construtor. Em seguida, utilizamos a função embutida len() para obter o tamanho da lista personalizada e armazenamos o resultado na variável tamanho.

Por fim, exibimos o tamanho da lista personalizada, que é o valor retornado pelo método __len__().

Função isinstance

A função isinstance() em Python é uma função embutida que permite verificar se um objeto pertence a uma determinada classe ou tipo de dados. Ela retorna True se o objeto for uma instância da classe especificada ou de qualquer uma de suas subclasses, caso contrário, retorna False.

A função isinstance() verifica se o objeto é uma instância direta ou indireta da classe fornecida. Isso significa que se o objeto for uma instância de uma subclasse da classe fornecida, a função também retornará True.

Acima, temos um exemplo de implementação na classe Ponto.

Exercícios para Praticar

Exercícios datetime

1. Crie um objeto datetime com a data e hora atuais e imprima-o.
2. Crie um objeto datetime com a data de nascimento do usuário e imprima-o.
3. Obtenha o dia da semana atual e imprima-o.
4. Obtenha a data atual e imprima-a no formato "DD/MM/AAAA".
5. Crie um objeto datetime com a data 31 de dezembro de 2022 e imprima-a.
6. Calcule a diferença de tempo entre a data atual e a data de nascimento do usuário e imprima-a em dias.
7. Crie um objeto datetime com a hora atual e imprima-a no formato de 12 horas.
8. Crie um objeto datetime com a data atual e adicione 5 dias a ela. Imprima a data resultante.
9. Crie um objeto datetime com a data atual e subtraia 2 semanas dela. Imprima a data resultante.
10. Crie um objeto datetime com a data atual e adicione 2 horas a ele. Imprima a hora resultante.
11. Crie um objeto datetime com a data e hora atuais. Imprima apenas a data.
12. Crie um objeto datetime com a data e hora atuais. Imprima apenas a hora.
13. Crie um objeto datetime com a data 1º de janeiro de 2022 e o horário 15:30. Imprima-o.
14. Converta a string "2022-05-18 12:30:00" em um objeto datetime e imprima-o.
15. Converta um objeto datetime em uma string no formato "AAAA-MM-DD HH:MM:SS" e imprima-a.
16. Crie um objeto datetime com a data atual e imprima o nome do mês.
17. Crie um objeto datetime com a data atual e imprima a data por extenso (exemplo: "18 de maio de 2022").
18. Crie um objeto datetime com a data atual e verifique se é um ano bissexto. Imprima o resultado.
19. Crie um objeto datetime com a data atual e obtenha o número do dia do ano. Imprima-o.
20. Crie um objeto datetime com a data atual e verifique se é um dia útil (segunda a sexta-feira). Imprima o resultado.
21. Crie um objeto datetime com a data 1º de janeiro de 2023 e verifique se é um dia útil. Imprima o resultado.
22. Crie um objeto datetime com a data atual e imprima o número da semana do ano.
23. Crie um objeto datetime com a data atual e verifique se é um ano bissexto. Imprima o resultado.
24. Crie um objeto datetime com a data atual e imprima o nome do dia da semana.
25. Crie um objeto datetime com a data atual e verifique se é o último dia do mês. Imprima o resultado.
26. Crie um objeto datetime com a data atual e verifique se é o último dia útil do mês. Imprima o resultado.
27. Crie um objeto datetime com a data atual e imprima a quantidade de dias restantes no mês.
28. Crie um objeto datetime com a data atual e verifique se é o horário de verão. Imprima o resultado.
29. Crie um objeto datetime com a data atual e verifique se é um feriado nacional. Imprima o resultado.
30. Crie um objeto datetime com a data atual e imprima a idade em anos, meses e dias.
31. Crie um objeto datetime com a data atual e imprima a hora atual no formato "HH:MM AM/PM".
32. Crie um objeto datetime com a data atual e calcule a diferença de tempo em relação a uma data específica fornecida pelo usuário. Imprima o resultado em dias.
33. Crie um objeto datetime com a data atual e verifique se é o horário de verão. Imprima o resultado.
34. Crie um objeto datetime com a data atual e verifique se é um dia de semana. Imprima o resultado.
35. Crie um objeto datetime com a data atual e verifique se é um dia feriado. Imprima o resultado.
36. Crie um objeto datetime com a data atual e adicione 3 horas e 30 minutos a ele. Imprima o resultado.
37. Crie um objeto datetime com a data atual e subtraia 2 horas e 15 minutos dele. Imprima o resultado.
38. Crie um objeto datetime com a data atual e verifique se é o primeiro dia do mês. Imprima o resultado.
39. Crie um objeto datetime com a data atual e verifique se é o último dia do ano. Imprima o resultado.
40. Crie um objeto datetime com a data atual e imprima a data e hora em um formato personalizado de sua escolha.

Exercícios Métodos Mágicos

41. Crie uma classe Ponto que represente um ponto no plano cartesiano. Implemente os métodos mágicos __add__(), __sub__(), e __mul__() para realizar operações aritméticas entre pontos.
42. Implemente uma classe Fracao que represente uma fração. Crie os métodos mágicos __add__(), __sub__(), __mul__(), e __truediv__() para realizar operações matemáticas com frações.
43. Crie uma classe Matriz que represente uma matriz 2x2. Implemente os métodos mágicos __add__(), __sub__(), e __mul__() para realizar operações aritméticas entre matrizes.
44. Implemente a classe Modulo que representa um número inteiro módulo n. Crie o método mágico __mod__() para calcular o resto da divisão entre dois objetos da classe Modulo.

45. Implemente a classe Texto que represente uma sequência de caracteres. Crie o método mágico __len__() para retornar o comprimento do texto.
46. Crie uma classe Data que represente uma data no formato dia/mês/ano. Implemente o método mágico __sub__() para calcular a diferença em dias entre duas datas.
47. Implemente a classe Intervalo que represente um intervalo de números inteiros. Crie o método mágico __len__() para retornar a quantidade de números no intervalo.
48. Crie uma classe ListaOrdenada que represente uma lista de elementos ordenados. Implemente o método mágico __add__() para concatenar duas listas ordenadas em uma nova lista ordenada.
49. Implemente a classe Tempo que represente um intervalo de tempo no formato horas:minutos:segundos. Crie o método mágico __add__() para somar dois objetos da classe Tempo e obter a soma total dos intervalos de tempo.
50. Crie uma classe Retangulo que represente um retângulo com altura e largura. Implemente os métodos mágicos __add__(), __sub__(), e __mul__() para calcular a área de retângulos.
51. Implemente a classe Palavra que represente uma palavra. Crie o método mágico __mul__() para repetir uma palavra várias vezes.
52. Crie uma classe Conjunto que represente um conjunto de elementos únicos. Implemente o método mágico __len__() para retornar a quantidade de elementos no conjunto.
53. Implemente a classe Velocidade que represente a velocidade de um objeto. Crie o método mágico __truediv__() para calcular o tempo necessário para percorrer uma determinada distância com a velocidade dada.
54. Crie uma classe Arquivo que represente um arquivo de texto. Implemente o método mágico __len__() para retornar a quantidade de caracteres no arquivo.
55. Crie uma classe ListaOrdenadaInvertida que represente uma lista de elementos ordenados em ordem decrescente. Implemente o método mágico __add__() para concatenar duas listas ordenadas invertidas em uma nova lista ordenada invertida.
56. Implemente a classe TempoDecorrido que represente o tempo decorrido desde um determinado evento. Crie o método mágico __sub__() para calcular a diferença de tempo entre dois objetos da classe TempoDecorrido.
57. Crie uma classe ConjuntoImutavel que represente um conjunto de elementos imutável. Implemente o método mágico __len__() para retornar a quantidade de elementos no conjunto.