Aula 08

Revisão

Mais alguns exemplos de uso:

```python
class Carro:
    """Classe Carro"""
    total_carros: int = 0

    def __init__(self, modelo: str, cor: str, ano: int) -> None:
        self.modelo: str = modelo
        self.cor: str = cor
        self.ano: int = ano
        self.pisca: bool = False
        Carro.total_carros += 1

    def buzina(self) -> None:
        """Buzina e verifica o estado do pisca do carro."""
        print(f'O carro {self.modelo} está buzinando.')
        print(
            f'e está com o pisca {"ligado" if self.pisca else "desligado"}.\n')

    def toggle_pisca(self) -> None:
        """Liga ou desliga o pisca do carro."""
        self.pisca = not self.pisca

    def __str__(self) -> str:
        return f'Carro: {self.modelo} - {self.cor} - {self.ano}'
```

```python
class Pessoa:
    """Definição da classe Pessoa"""
    lista_pessoas: list = []

    def __init__(self, p_nome: str, p_idade: int) -> None:
        self.nome: str = p_nome
        self.idade: int = p_idade
        self.ja_nasceu: bool = True

    def apresenta(self) -> None:
        """Método para mostrar a saudação da pessoa"""
        print(f'Olá, eu sou {self.nome} e tenho {self.idade} anos.')

    def __str__(self) -> str:
        return f'Pessoa: {self.nome} - {self.idade} anos.'
```

```python
class Calculadora:
    """Classe Calculadora"""
    lista_operacoes: list = []

    def __init__(self, p_valor1: int | float, p_valor2: int | float) -> None:
        self.valor_1: int | float = p_valor1
        self.valor_2: int | float = p_valor2

    def soma(self) -> None:
        """Soma os valores"""
        soma: int | float = self.valor_1 + self.valor_2
        print(f'A soma de {self.valor_1} + {self.valor_2} é igual a: {soma}')
        calculo: dict = {}
        calculo['valor_1'] = self.valor_1
        calculo['operacao'] = '+'
        calculo['valor_2'] = self.valor_2
        calculo['resultado'] = soma
        Calculadora.lista_operacoes.append(calculo)

    def __str__(self) -> str:
        return f'{Calculadora.lista_operacoes}'
```

```python
if __name__ == '__main__':
    # instanciando um novo objeto e com anotações de tipo
    p1: Pessoa = Pessoa('Arnold Schwarzenegger', 75)
    p1.apresenta()

    calc: Calculadora = Calculadora(10, 20)
    print(calc)
    calc.soma()
    print(calc)
```

Mais Funções

Funções max e min

A função max() e min() são funções embutidas do Python que são usadas para encontrar o valor máximo e mínimo em um conjunto de valores, respectivamente. Elas podem ser aplicadas a diferentes tipos de dados, como números, sequências e objetos.

A função max() retorna o maior valor entre os argumentos fornecidos. Você pode fornecer vários argumentos separados por vírgulas ou um iterador. Aqui está um exemplo simples:

```python
lista: list = [10, 13, 41, 1, 23, 40]
maior: int = max(lista)
print(maior)
```

No exemplo acima, a função max() retorna o maior número da lista, que é 41.

A função min() retorna o menor valor entre os argumentos fornecidos. Neste caso, o menor número é 1.

```python
lista: list = [10, 13, 41, 1, 23, 40]
menor: int = min(lista)
print(menor)
```

As funções max() e min() também podem ser aplicadas a sequências, como listas ou tuplas. Nesse caso, elas retornarão o elemento máximo ou mínimo com base na ordem lexicográfica (ordem alfabética para strings e ordem crescente para números). Veja um exemplo:

```python
numeros: list[int] = [10, 13, 41, 1, 23, 40]
palavras: list[str] = ['Python', 'Java', 'C++', 'C#', 'JavaScript']
menor: int = min(numeros)
maior: str = max(palavras)
print(menor)
print(maior)
```

No exemplo acima, max() retorna o maior número da lista numeros, enquanto min() retorna a primeira palavra em ordem alfabética da lista palavras.

Função enumerate

A função enumerate() é uma função embutida do Python que permite iterar sobre uma sequência (como uma lista, tupla, string, etc.) enquanto simultaneamente acompanha o índice de cada elemento. Ela retorna um objeto enumerado que contém pares de valores, onde cada par consiste em um índice e o elemento correspondente da sequência.

A sintaxe básica da função enumerate() é a seguinte:

```
enumerate(iterable, start=0)
```

O parâmetro iterable é a sequência na qual você deseja iterar, e o parâmetro opcional start especifica o valor inicial do índice. Por padrão, o valor inicial é 0, mas você pode definir qualquer valor inteiro desejado.

A função enumerate() retorna um objeto de tipo iterador que gera tuplas contendo o índice e o elemento em cada iteração. Para acessar o índice e o elemento em cada iteração, você pode desempacotar a tupla retornada pelo iterador ou acessá-los diretamente.

Aqui está um exemplo simples para ilustrar como a função enumerate() funciona:

```
linguagens: list[str] = ['Python', 'Java', 'C++', 'C#', 'JavaScript']
for indice, linguagem in enumerate(linguagens):
    print(f'Índice: {indice} - Linguagem: {linguagem}')
```

A saída do exemplo acima seria:

```
Índice: 0 - Linguagem: Python
Índice: 1 - Linguagem: Java
Índice: 2 - Linguagem: C++
Índice: 3 - Linguagem: C#
Índice: 4 - Linguagem: JavaScript
```

No exemplo acima, a função enumerate(linguagens) retorna um objeto iterador que gera as seguintes tuplas em cada iteração: (0, "Python"), (1, "Java"), (2, "C++"), etc. O desempacotamento das tuplas ocorre na declaração indice, linguagem, onde indice recebe o valor do índice e linguagem recebe o valor do elemento correspondente da lista linguagens. Dentro do loop, esses valores são impressos.

A função enumerate() é útil quando você precisa iterar sobre uma sequência e ao mesmo tempo rastrear o índice dos elementos. Isso é especialmente útil em situações em que você precisa modificar ou acessar elementos com base no índice enquanto itera.

Herança

A herança é um conceito fundamental na programação orientada a objetos (POO) e permite a criação de novas classes baseadas em classes existentes. No Python, a herança é implementada por meio da declaração da classe filha (ou subclasse) que herda características e comportamentos da classe pai (ou superclasse).

A sintaxe básica para definir uma classe filha que herda da classe pai em Python é a seguinte:

```
class ClassePai:
    """Definição da classe pai"""

class ClasseFilha(ClassePai):
    """Definição da classe filha que herda da classe pai"""
```

Na classe filha, podemos adicionar novos atributos e métodos ou modificar os existentes. Ela terá acesso a todos os atributos e métodos públicos da classe pai.

Quando você cria um objeto da classe filha, ele pode acessar tanto os atributos e métodos definidos na classe filha quanto os herdados da classe pai. Se um atributo ou método é chamado em uma instância da classe filha, o Python verifica primeiro na própria classe filha. Se não for encontrado lá, ele verifica na classe pai e em qualquer outra classe ancestral na hierarquia de herança.

A herança também permite a sobreposição de métodos (também conhecido como override), o que significa que você pode definir um método com o mesmo nome tanto na classe pai quanto na classe filha. Quando o método é chamado em uma instância da classe filha, o método da classe filha é executado em vez do método da classe pai.

O Python suporta duas formas de herança: simples e múltipla. Veremos herança múltipla nas próximas aulas.

Herança Simples

A característica de herança simples é ela herdar apenas uma classe.
Veja o exemplo abaixo:

```python
class Animal:
    """Classe Animal"""

    def __init__(self, p_nome: str) -> None:
        self.nome: str = p_nome

    def som(self) -> None:
        """Método para emitir som do animal"""
        print('O animal está emitindo som ... ')

    def __str__(self) -> str:
        return f'Animal: {self.nome}'
```

```python
class Cao(Animal):
    """Classe Cao que herda da classe Animal"""

    def __init__(self, p_nome: str, p_cor_pelo: str) -> None:
        super().__init__(p_nome)
        self.cor_pelo: str = p_cor_pelo

    def som(self) -> None:
        """Método para emitir som do cachorro"""
        print('O cachorro está latindo ... ')

    def __str__(self) -> str:
        return f'Cachorro: {self.nome}'
```

```python
class Ave(Animal):
    """Classe Ave que herda da classe Animal"""

    def __init__(self, p_nome: str, p_cor_pena: str) -> None:
        super().__init__(p_nome)
        self.cor_pena: str = p_cor_pena

    def som(self) -> None:
        """Método para emitir som da ave"""
        print('A ave está cantando ... ')

    def __str__(self) -> str:
        return f'Ave: {self.nome}'
```

```python
class Gato(Animal):
    """Classe Gato que herda da classe Animal"""

    def som(self) -> None:
        """Método para emitir som do gato"""
        print('O gato está miando ... ')

    def arranha(self) -> None:
        """Método para o gato arranhar"""
        print('O gato está arranhando ... ')

    def __str__(self) -> str:
        return f'Gato: {self.nome}'
```

Uma vez criada as classes, vamos instanciar os objetos e chamar seus métodos.

```python
"""módulo main.py"""
from classes import Animal, Cao, Gato, Ave

# Instanciando objetos
generico = Animal('Genérico')
rex = Cao('Rex', 'Marrom')
cocota = Ave('Cocota', 'Verde')
garfield = Gato('Garfield')

# Chamando métodos
generico.som()
rex.som()
cocota.som()
garfield.som()
garfield.arranha()
print()

# Imprimindo objetos
print(generico)
print(rex)
print(cocota)
print(garfield)
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python main.py
O animal está emitindo som ...
O cachorro está latindo ...
A ave está cantando ...
O gato está miando ...
O gato está arranhando ...

Animal: Genérico
Cachorro: Rex
Ave: Cocota
Gato: Garfield

PS C:\Users\gutoh\OneDrive\0.Python>
```

Reparou a falta de um construtor (método __init__()) na classe Gato? Isso acontece porque ele não possui implementação de novos atributos (como a cor_pelo e cor_pena), então não é necessário recriar ele. Em compensação, a classe Gato é a única a ter a criação deu um método a mais (arranha()).

Função super()

A função super() é uma função embutida em Python que é frequentemente usada dentro do método __init__ de uma classe filha para chamar o método __init__ da classe pai. Ela permite que a classe filha herde e inicialize os atributos da classe pai antes de adicionar seus próprios atributos específicos.

Quando uma classe filha herda de uma classe pai, ela pode substituir ou adicionar comportamentos aos métodos da classe pai. No entanto, às vezes, é necessário executar o código do método da classe pai antes de adicionar a funcionalidade específica da classe filha. É aí que a função super() é útil.

Ao chamar super().__init__(), a classe filha chama o método __init__ da classe pai, passando os mesmos argumentos que recebeu. Isso permite que a classe pai inicialize seus próprios atributos antes que a classe filha possa fazer qualquer coisa adicional.

Exercícios para Praticar

Exercícios min e max

1. Dada uma lista de números [3, 7, 2, 9, 1], use a função min() para encontrar o valor mínimo.
2. Dada uma lista de palavras ["banana", "maçã", "laranja"], use a função max() para encontrar a palavra com o maior número de letras.
3. Dada uma string "python", use a função min() para encontrar o caractere com o valor Unicode mínimo.
4. Dada uma lista de números [-5, 0, 12, -3, 8], use a função max() para encontrar o valor máximo absoluto.
5. Dada uma lista de números [1, 2, 3, 4, 5], use a função min() para encontrar o valor mínimo e divida-o por 2.
6. Dada uma lista de strings ["abc", "defg", "hijkl"], use a função max() para encontrar a string com o maior número de caracteres.
7. Dada uma lista de números [7, 3, 9, 2, 1], use a função min() para encontrar o segundo menor valor.
8. Dada uma lista de tuplas [(1, 5), (3, 2), (4, 8), (2, 7)], use a função max() para encontrar a tupla com o maior valor no segundo elemento.
9. Dada uma lista de listas [[1, 2, 3], [4, 5, 6], [7, 8, 9]], use a função min() para encontrar a lista com o menor valor máximo.
10. Dada uma lista de dicionários [{"nome": "João", "idade": 25}, {"nome": "Maria", "idade": 30}, {"nome": "Pedro", "idade": 20}], use a função max() para encontrar o dicionário com o maior valor de idade.
11. Dada uma lista de números [2, 4, 6, 8, 10], use a função min() para encontrar o primeiro número par.
12. Dada uma lista de strings ["abc", "def", "ghi", "jkl"], use a função max() para encontrar a string com o maior valor Unicode.
13. Dada uma lista de números [5, 10, 15, 20, 25], use a função min() para encontrar o valor mínimo e multiplique-o por 3.
14. Dada uma lista de dicionários [{"nome": "João", "idade": 25}, {"nome": "Maria", "idade": 30}, {"nome": "Pedro", "idade": 20}], use a função max() para encontrar o dicionário com o maior valor de idade e imprima apenas o nome.
15. Dada uma lista de strings ["apple", "banana", "cherry", "date"], use a função min() para encontrar a string com o menor número de caracteres.
16. Dada uma lista de números [5, 10, 3, 8, 1], use a função max() para encontrar o segundo maior valor.
17. Dada uma lista de tuplas [(3, 6), (2, 8), (5, 4), (1, 9)], use a função min() para encontrar a tupla com o menor valor no segundo elemento.
18. Dada uma lista de listas [[10, 20, 30], [40, 50, 60], [70, 80, 90]], use a função max() para encontrar a lista com o maior valor mínimo.
19. Dada uma lista de dicionários [{"nome": "João", "idade": 25}, {"nome": "Maria", "idade": 30}, {"nome": "Pedro", "idade": 20}], use a função min() para encontrar o dicionário com o menor valor de idade.
20. Dada uma lista de números [1, 2, 3, 4, 5], use a função max() para encontrar o valor máximo e verifique se ele é igual a 5.

Exercícios enumerate

21. Dada uma lista de números [3, 7, 2, 9, 1], use a função enumerate() para imprimir o índice e o valor de cada elemento.
22. Dada uma string "python", use a função enumerate() para imprimir o índice e o caractere correspondente de cada letra.
23. Dada uma lista de palavras ["banana", "maçã", "laranja"], use a função enumerate() para encontrar o índice da palavra "maçã".
24. Dada uma lista de números [10, 20, 30, 40, 50], use a função enumerate() para encontrar o índice do primeiro número maior que 25.
25. Dada uma lista de nomes ["João", "Maria", "Pedro"], use a função enumerate() para criar um dicionário onde as chaves são os índices e os valores são os nomes correspondentes.
26. Dada uma lista de cores ["vermelho", "azul", "verde"], use a função enumerate() para imprimir apenas as cores nos índices pares.
27. Dada uma lista de números [1, 4, 7, 3, 6], use a função enumerate() para imprimir apenas os números nos índices ímpares.
28. Dada uma lista de palavras ["olá", "mundo", "python"], use a função enumerate() para imprimir as palavras em ordem reversa, junto com seus índices correspondentes.
29. Dada uma lista de números [5, 10, 15, 20], use a função enumerate() para calcular a soma de cada número multiplicado pelo seu índice.

30. Dada uma lista de frutas ["maçã", "banana", "laranja"], use a função enumerate() para imprimir as frutas em ordem alfabética, juntamente com seus índices correspondentes.
31. Dada uma lista de nomes ["João", "Maria", "Pedro"], use a função enumerate() para criar uma nova lista contendo apenas os nomes que começam com a letra "P".
32. Dada uma lista de números [2, 5, 8, 11, 14], use a função enumerate() para encontrar o índice do primeiro número divisível por 3.
33. Dada uma lista de strings ["abc", "def", "ghi"], use a função enumerate() para imprimir cada string juntamente com o seu índice e o número total de caracteres.
34. Dada uma lista de números [3, 6, 9, 12, 15], use a função enumerate() para encontrar o índice do último número divisível por 4.
35. Dada uma lista de frutas ["maçã", "banana", "laranja"], use a função enumerate() para criar uma nova lista contendo tuplas com o nome da fruta e seu índice correspondente.
36. Dada uma lista de nomes ["João", "Maria", "Pedro"], use a função enumerate() para imprimir apenas os nomes nos índices pares.
37. Dada uma lista de números [10, 20, 30, 40], use a função enumerate() para encontrar o índice do último número maior que 25.
38. Dada uma string "python", use a função enumerate() para imprimir cada caractere junto com seu índice reverso.
39. Dada uma lista de cores ["vermelho", "azul", "verde"], use a função enumerate() para criar um dicionário onde as chaves são as cores e os valores são os índices correspondentes.
40. Dada uma lista de números [1, 2, 3, 4, 5], use a função enumerate() para encontrar o índice do primeiro número igual a ele mesmo.

Exercícios Herança

41. Crie uma classe chamada Veiculo com um método mover que imprima "O veículo está em movimento".
    a. Crie uma classe chamada Carro que herde da classe Veiculo. Adicione um método adicional chamado abrir_porta que imprima "A porta do carro está aberta".
    b. Crie uma instância da classe Carro e chame o método mover e abrir_porta.
    c. Crie uma classe chamada Moto que herde da classe Veiculo. Adicione um método adicional chamado acelerar que imprima "A moto está acelerando".
    d. Crie uma instância da classe Moto e chame os métodos mover e acelerar.
42. Crie uma classe chamada Animal com um método emitir_som que imprima "O animal emite um som".
    a. Crie uma classe chamada Cachorro que herde da classe Animal. Adicione um método adicional chamado abanar_rabo que imprima "O cachorro está abanando o rabo".
    b. Crie uma instância da classe Cachorro e chame os métodos emitir_som e abanar_rabo.
    c. Crie uma classe chamada Gato que herde da classe Animal. Adicione um método adicional chamado arranhar que imprima "O gato está arranhando".
    d. Crie uma instância da classe Gato e chame os métodos emitir_som e arranhar.
43. Crie uma classe chamada Aluno com um método estudar que imprima "O aluno está estudando".
    a. Crie uma classe chamada Estudante que herde da classe Aluno. Adicione um método adicional chamado fazer_tarefa que imprima "O estudante está fazendo a tarefa".
    b. Crie uma instância da classe Estudante e chame os métodos estudar e fazer_tarefa.
    c. Crie uma classe chamada Professor com um método ensinar que imprima "O professor está ensinando".
    d. Crie uma classe chamada ProfessorUniversitario que herde da classe Professor. Adicione um método adicional chamado realizar_pesquisa que imprima "O professor universitário está realizando uma pesquisa".
    e. Crie uma instância da classe ProfessorUniversitario e chame os métodos ensinar e realizar_pesquisa.
44. Crie uma classe chamada Forma com um método calcular_area que retorne 0.
    a. Crie uma classe chamada Quadrado que herde da classe Forma. Sobrescreva o método calcular_area para calcular a área de um quadrado.
    b. Crie uma instância da classe Quadrado e chame o método calcular_area. Verifique se a área calculada está correta.
    c. Crie uma classe chamada Circulo que herde da classe Forma. Sobrescreva o método calcular_area para calcular a área de um círculo.
    d. Crie uma instância da classe Circulo e chame o método calcular_area. Verifique se a área calculada está correta.

45. Crie uma classe chamada Pessoa com um atributo nome e um método apresentar que imprima "Olá, meu nome é <nome>".
    a. Crie uma classe chamada Estudante que herde da classe Pessoa. Adicione um atributo curso e um método estudar que imprima "Estou estudando <curso>".
    b. Crie uma instância da classe Estudante e chame os métodos apresentar e estudar.
46. Crie uma classe chamada Funcionario com um atributo cargo e um método trabalhar que imprima "Estou trabalhando como <cargo>".
    a. Crie uma classe chamada Gerente que herde da classe Funcionario. Adicione um método adicional chamado liderar_equipe que imprima "Estou liderando a equipe".
    b. Crie uma instância da classe Gerente e chame os métodos trabalhar e liderar_equipe.
47. Crie uma classe chamada ContaBancaria com atributos titular e saldo. Adicione um método depositar que atualize o saldo e um método sacar que reduza o saldo.
    a. Crie uma classe chamada ContaCorrente que herde da classe ContaBancaria. Adicione um atributo adicional limite_cheque_especial e um método usar_cheque_especial que permita que o titular da conta use o limite do cheque especial.
    b. Crie uma instância da classe ContaCorrente e teste os métodos depositar, sacar e usar_cheque_especial.
48. Crie uma classe chamada Animal com um método comer que imprima "O animal está comendo".
    a. Crie uma classe chamada Cavalo que herde da classe Animal. Adicione um método adicional chamado correr que imprima "O cavalo está correndo".
    b. Crie uma instância da classe Cavalo e chame os métodos comer e correr.
49. Crie uma classe chamada Shape com um método calcular_area que retorne 0.
    a. Crie uma classe chamada Retangulo que herde da classe Shape. Adicione atributos largura e altura e sobrescreva o método calcular_area para calcular a área do retângulo.
    b. Crie uma instância da classe Retangulo e chame o método calcular_area. Verifique se a área calculada está correta.
    c. Crie uma classe chamada Triangulo que herde da classe Shape. Adicione atributos base e altura e sobrescreva o método calcular_area para calcular a área do triângulo.
    d. Crie uma instância da classe Triangulo e chame o método calcular_area. Verifique se a área calculada está correta.
50. Crie uma classe chamada Fruta com um método descascar que imprima "Descascando a fruta".
    a. Crie uma classe chamada Laranja que herde da classe Fruta. Adicione um método adicional chamado espremer que imprima "Espremendo a laranja".
    b. Crie uma instância da classe Laranja e chame os métodos descascar e espremer.
51. Crie uma classe chamada Veiculo com um método ligar que imprima "O veículo está ligado".
    a. Crie uma classe chamada Carro que herde da classe Veiculo. Adicione um método adicional chamado dirigir que imprima "Dirigindo o carro".
    b. Crie uma instância da classe Carro e chame os métodos ligar e dirigir.
    c. Crie uma classe chamada Aviao que herde da classe Veiculo. Adicione um método adicional chamado voar que imprima "O avião está voando".
    d. Crie uma instância da classe Aviao e chame os métodos ligar e voar.
52. Crie uma classe chamada Conta com atributos numero e saldo. Adicione um método depositar que atualize o saldo e um método sacar que reduza o saldo.
    a. Crie uma classe chamada ContaPoupanca que herde da classe Conta. Adicione um atributo adicional juros e um método calcular_juros que calcule o juros do saldo.
    b. Crie uma instância da classe ContaPoupanca e teste os métodos depositar, sacar e calcular_juros.
53. Crie uma classe chamada Pessoa com atributos nome e idade. Adicione um método apresentar que imprima "Eu sou <nome> e tenho <idade> anos".
    a. Crie uma classe chamada Aluno que herde da classe Pessoa. Adicione um atributo adicional curso e um método estudar que imprima "Estou estudando <curso>".