



devaria

Programação – Python

Parte 2

Parte 1

- Estrutura de repetição

Estrutura de Repetição

Agora vamos conhecer as Estrutura de repetição da linguagem. Estas são responsáveis por executarem as instruções definidas no escopo de sua estrutura diversas vezes até que uma condição seja atingida. As principais estruturas de repetição que temos no Python são:

- for (faça/para);
- while (enquanto);



Estrutura de repetição

for – Uma das possibilidades com o for é criar repetições de acordo com um intervalo de valores

Útil para quando sabemos previamente a quantidade de vezes que precisamos iterar

```
for n in range(0, 3):  
    print(n)
```



python

Estrutura de repetição

for – É possível também utilizar de forma decrescente

```
for n in range(3, 0, -1):  
    print(n)
```



python



Estrutura de repetição

for – Podemos utilizar também para percorrer os caracteres de uma string

```
palavra = "devaria"
```

```
for letra in palavra:  
    print(letra)
```



python



Estrutura de repetição

for – Podemos utilizar também para percorrer objetos em uma lista

```
pessoas = ["Filipe", "Daniel", "Rafael", "Douglas"]
```

```
for p in pessoas:  
    darBoasVindas(p)
```



python

Estrutura de repetição

for – Conseguimos também acessar o index dos elementos utilizando a função enumerate

```
pessoas = ["Filipe", "Daniel", "Rafael", "Douglas"]
```

```
for i, p in enumerate(pessoas, start=0):  
    print(f'Pessoa: {p} no Index: {i}')
```



python



Estrutura de repetição

while – Verifica a condição e se for verdadeira a condição as instruções dentro do escopo são executadas a cada novo loop.

Útil para quando a condição de parada não depende de um número

```
temCarroNaFila = True
while temCarroNaFila:
    lavarCarro()
    temCarroNaFila = verificarTemCarroNaFila()
```



python

Estrutura de repetição

while – É possível também utilizar um else no while para quando ele sair do While executar algo específico

```
temCarroNaFila = True
while temCarroNaFila:
    lavarCarro()
    temCarroNaFila = verificarTemCarroNaFila()
else:
    print("Terminou de lavar os carros")
```



python

```
11
12
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)
14
15
16 class Base(ContainerAware, metaclass=abc.ABCMeta):
17     """The base class for all controllers.
18     Attributes:
19         __action__ (string): The last action that was
20         executed.
21     """
22     def execute(self, **kwargs):
23         method = self.get_execute_method(**kwargs)
24         self.__action__ = method.__name__
25         return method(**kwargs) or {}
```

Estrutura de repetição

Agora que já entendemos como funcionam as estruturas de repetição. Nesse exercícios vamos exercitar algumas coisas:

- Criar um novo programa;
- Capturar as informações passadas nos argumentos;
- Testar todas as formas de estrutura de repetição;
- Utilizar a depuração do PyCharm;
- Retornar para o usuário tudo que foi digitado na linha de comando;



python



Café ou Dúvidas?

Para que a aula não fique cansativa, a cada conclusão de assunto teremos 15 minutos para que vocês possam tirar as dúvidas através dos comentários no vídeo e responderemos ao vivo.

Parte 2

 Tratamento de Exceção

 Listas

 Funções anônimas

Tratamento de Exceção

Um dos principais pontos de desenvolvimento e que fazemos desde o nosso primeiro programa é evitar que nosso usuário tenha mensagens e retornos muito técnicas que ele não faz a menor ideia do que está sendo dito, além disso nossos programas são vivos e dinâmicos o que acaba tornando muito difícil mapear todos os possíveis erros que nosso programa poderá ocorrer.

Quando começarmos a desenvolver um sistema mesmo será muito trabalhoso prever todas as nuances que o sistema poderá se comportar e seria inevitável que algum comportamento desse um erro muito técnico para nosso usuário ex. “Cannot connect to a database”

Para esses casos temos uma abordagem muito elegante e pratica que são os tratamentos de exceção:

- try/except (tentar/exceto);



Try / Except

try/except – essa abordagem abre um escopo onde, tudo que for executado dentro daquele escopo caso gere um erro o escopo do except será iniciado e o escopo do try para de executar

Usuarios = **None**

try:

 usuarios = buscarTodosUsuarios()

except Exception as e:

print(f“Ocorreu um erro ao buscar os usuários”)



python

Listas

Agora vamos falar um pouco mais sobre as listas no Python.

Como já explicamos listas/arrays são variáveis que ao invés de guardar um único dado guardam uma série de dados do mesmo tipo.

São muito úteis, pois guardam todas as linhas de uma tabela, ou uma lista de nomes credenciados, ou uma lista de dígitos válidos para validar uma informação.

```
notas = [ 9, 7, 8, 93 7, 8 ]
```

```
funcionarios = [ "Emerson", "Zacarias", "Morgana" ]
```



python

Listas

Manipular listas não é uma tarefa fácil, elas são muito utilizadas no dia-a-dia e precisamos sempre exercer inteligência nelas como:

- Encontrar um registro no meio da lista que atenda uma condição;
- Separar a lista em listas menores;
- Ordenar a lista de acordo com um atributo;
- Formatar a lista para facilitar a manipulação;

No caso do Python, essa manipulação se dá uma forma relativamente simples e compreensível. A própria classe de lista do Python dispõe de **métodos** que nos auxiliam nesse trabalho.



python

Listas

Entre os principais métodos estão:

- `.append(x)` -> Adiciona um item 'x' no final lista;
- `.insert(i, x)` -> Adiciona um item 'x' na posição 'i' da lista;
- `.remove(x)` -> Remove um item 'x' da lista;
- `.sort()` -> Ordena a lista de acordo com a condição/atributo passado;
- `.clear()` -> Remove todos os itens da lista;
- `.count(x)` -> Retorna a quantidade de vezes que o item 'x' aparece na lista



python

Funções anônimas

As funções anônimas são rotinas definidas sem um identificador as associando e são geralmente criadas para implementar pequenas funcionalidades ou para serem utilizadas por um período breve (dentro do escopo de uma função, por exemplo).

As funções anônimas no Python são definidas com a seguinte sintaxe:

`lambda <<argumentos>> : <<expressão>>`



Funções anônimas

Fazer uma média das notas que estão salva em uma lista.

```
notasDosAlunos = [ 90, 71, 82, 93, 75, 82 ]
```

```
mediaDasNotas = lambda notas : sum(notas) / len(notas)
```

```
print(f'A média das notas é {mediaDasNotas(notasDosAlunos)}')
```



python

Funções anônimas

Função anônima:

```
mediaDasNotas = lambda notas : sum(notas) / len(notas)
```

É equivalente a seguinte função:

```
def mediaDasNotas(notas):  
    media = sum(notas) / len(notas)  
  
    return media
```



python



Funções anônimas

Filtrar notas acima de 80.

```
notas_alunos = [ 90, 71, 82, 93, 75, 82 ]
```

```
notas_filtradas = list(filter(lambda i : i > 80, notas_alunos))
```

```
print(f'Notas acima de 80 {notas_filtradas}')
```



python

```
11
12
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)
14
15
16 class Base(ContainerAware, metaclass=abc.ABCMeta):
17     """The base class for all controllers.
18     Attributes:
19         __action__ (string): The last action that was
20         executed.
21     """
22     def execute(self, **kwargs):
23         method = self.get_execute_method(**kwargs)
24         self.__action__ = method
25         return method(**kwargs) or {}
```


Funções anônimas

Função anônima:

```
notas_filtradas = list(filter(lambda i : i > 80, notas_alunos))
```

É equivalente a seguinte função:

```
def filtrar_notas(lista_notas):  
    notas_filtradas = []  
  
    for nota in lista_notas:  
        if nota > 80:  
            notas_filtradas.append(nota)  
  
    return notas_filtradas
```



Hora da prática

Agora que já entendemos como funcionam exceções, listas e funções anônimas. Nesse exercícios vamos exercitar algumas coisas:

- Criar um novo programa;
- Tratar nossas possíveis exceções;
- Percorrer listas e arrays;
- Utilizar métodos do array para Validar/Ordernar a lista;
- Retornar para o usuário o resultado do programa;



python

Hora da prática

Desafio:

Escrever um programa que recebe alguns produtos como argumento, validar se esse produtos estão na lista de itens disponíveis do mercado, caso esteja avisar o usuário quais produtos tem e quais não tem e por ultimo exibir a lista de produtos disponíveis ordenados por ordem alfabética do mercado para que o usuário possa pedir na próxima vez.








Café ou Dúvidas?

Para que a aula não fique cansativa, a cada conclusão de assunto teremos 15 minutos para que vocês possam tirar as dúvidas através dos comentários no vídeo e responderemos ao vivo.

Parte 3

-  Classes e Objetos
-  Atributos e Métodos de Classe
-  Construtores

Classes

Como já falamos na linguagem de programação, classes são representações do mundo real abstraídas para o programa/sistema a fim de utilizar suas características ou comportamentos dentro das instruções do código.

Classes são o elemento principal da orientação a objeto onde toda a regra de negócio do programa/sistema estará.

Vale lembrar que a classe é o MOLDE daquilo que queremos usar no programa/sistema e que esse molde não precisa conter exatamente tudo que ele tem no mundo real, utilizaremos somente aquilo necessário para que o problema a qual o programa/sistema resolve seja atendido



python

Classes

classe – uma classe é sempre iniciada pela palavra chave class com um nome que identifica essa classe, após isso cria-se o escopo onde tudo que estiver nele pertence a classe, seja atributos, sejam métodos:

```
class ContaBancaria:
```



python



Objetos

Já os objetos são o preenchimento dos nossos moldes salvos na memória do programa, ou seja, criamos uma variável do tipo daquela Classe, preenchemos os valores de acordo com os dados passados e na sequência salvamos na memória aquele dado para ser utilizado no futuro.

A grande vantagem de utilizar objetos ao invés de múltiplas variáveis é justamente a facilidade de manipulação e sentido dentro do programa, facilitando o programador entender que todos aqueles infinitos dados pertencem a uma classe/molde que faz sentido no mundo real.



python

```
11
12
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)
14
15
16 class Base(ContainerAware, metaclass=abc.ABCMeta):
17     """The base class for all controllers.
18     Attributes:
19         __action__ (string): The last action that was
20         executed.
21     """
22     def execute(self, **kwargs):
23         method = self.get_execute_method(**kwargs)
24         self.__action__ = method
25         return method(**kwargs) or {}
```

Objetos

objetos – cria-se uma variável para guardar o objeto, após isso atribui-se = uma nova INSTÂNCIA daquela Classe, assim como as funções tem parâmetros de entrada, as classes tem construtores que podem ou não receber dados para construir um objeto:

```
conta1 = ContaBancaria()  
conta2 = ContaBancaria()  
conta3 = ContaBancaria()
```



python

Construtores

Construtor é a função que é executada ao instanciar uma classe, normalmente tem a responsabilidade de inicializar alguns atributos importantes para funcionamento das mesmas.

Outra função importante dos construtores é justamente não permitir criar objetos sem que os dados obrigatórios para preenchimento sejam passados, evitando erros no programa.



python

Construtores

`__init__` – Método reservado para iniciar o construtor, o parâmetro `self` é obrigatório

```
class ContaBancaria:  
    def __init__(self):  
        pass
```



python

```
11  
12  
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)  
14  
15  
16 class Base(ContainerAware, metaclass=abc.ABCMeta):  
17     """The base class for all controllers.  
18     Attributes:  
19         __action__ (string): The last action that was  
20         executed.  
21     """  
22     def execute(self, **kwargs):  
23         method = self.get_execute_method(**kwargs)  
24         self.__action__ = method  
25         return method(**kwargs) or {}
```

Atributos/Propriedades

São as características que aquele molde deverá ter para atender o problema a qual o programa resolve.

São variáveis, porém essas variáveis PERTENCEM a classe e só podem ser acessadas quando o objeto está instanciado.

Tanto atributos quanto métodos são acessados através do operador “.” que navega para dentro da classe e identifica a lista de atributos e métodos definidos para aquela classe



python

Atributos/Propriedades

// atributos – No Python, geralmente definimos as propriedades diretamente no construtor

```
class ContaBancaria:
    def __init__(self, tipo, numero, agencia):
        self.tipo = tipo
        self.numero = numero
        self.agencia = agencia
    }
```



python

```
11
12
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)
14
15
16 class Base(ContainerAware, metaclass=abc.ABCMeta):
17     """The base class for all controllers.
18     Attributes:
19         __action__ (string): The last action that was
20         executed.
21     """
22     def execute(self, **kwargs):
23         method = self.get_execute_method(**kwargs)
24         self.__action__ = method
25         return method(**kwargs) or {}
```

Atributos

agora vamos instancia um objeto para atribuir valor aos atributos e acessa-los

```
conta1 = ContaBancaria("corrente", 35662, "0001")
```

```
print(f"Conta criada, tipo: {conta1.tipo}, agencia: {conta1.agencia} e numero: {conta1.numero}")
```

```
conta2 = ContaBancaria("poupança", 214544, "0203")
```

```
print("Conta criada, tipo: {conta2.tipo}, agencia: {conta2.agencia} e numero: {conta2.numero}")
```



python

Métodos

São as ações/comportamentos que aquele molde deverá ter para atender o problema a qual o programa resolve.

São funções, porém essas funções PERTENCEM a classe ou a instância (objeto).

Seguem o mesmo padrão das funções que temos no código, nome, parâmetros de entrada e escopo. Uma boa vantagem aqui é que elas conseguem acessar os atributos da instancia do objeto sem restrição diretamente.



python

Métodos

atributos – primeiro definimos a ação que a classe terá no programa

```
class ContaBancaria:
    // ...restante da classe
    def exibirDadosConta(self):
        print("Conta, tipo: {self.tipo},
            agencia: {self.agencia} e numero: {self.numero}")
    }
}
```



python

```
11
12
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)
14
15
16 class Base(ContainerAware, metaclass=abc.ABCMeta):
17     """The base class for all controllers.
18     Attributes:
19         __action__ (string): The last action that was
20         executed.
21     """
22     def execute(self, **kwargs):
23         method = self.get_execute_method(**kwargs)
24         self.__action__ = method
25         return method(**kwargs) or {}
```

Métodos

// agora com a instancia dos objetos vamos utilizar a função para exibição

```
conta1 = ContaBancaria("Corrente", 256351, "0001")
```

```
conta1.exibirDadosConta()
```



python



Hora da prática

Agora que já entendemos como funcionam classes, objetos, construtores, atributos e métodos. Nesse exercício vamos exercitar algumas coisas:

- Criar um novo programa;
- Criar uma classe;
- Transformar nossos argumentos em objetos;
- Retornar para o usuário o resultado do programa;



python

Hora da prática

Desafio (só que agora com classes =D):

Escrever um programa que recebe alguns produtos como argumento, validar se esse produtos estão na lista de itens disponíveis do mercado, caso esteja avisar o usuário quais produtos tem e quais não tem e por ultimo exibir a lista de produtos disponíveis ordenados por ordem alfabética do mercado para que o usuário possa pedir na próxima vez.



python

Não se esqueça de commitar
Os programas desenvolvidos no seu Git =D





Obrigado pela participação

Esperamos que você tenha gostado, fique à vontade para nos enviar seus feedbacks sobre esta aula.
Esperamos você na próxima live, segunda-feira 10/05/21 às 19:30. Não se esqueça de consultar o calendário e anote na sua agenda.

Se inscreva no canal e siga-nos nas rede sociais:



www.youtube.com/c/Devaria



[@devaria_oficial](https://www.instagram.com/devaria_oficial)

[@rafamazucato](https://www.instagram.com/rafamazucato)

[@castellodaniel](https://www.instagram.com/castellodaniel)

[@oliveirandouglas](https://www.instagram.com/oliveirandouglas)