Workshop Python Basic {

[Aula 2]

```
    print(“Funções, Estruturas de dados
    e estruturas de repetição”)
```

}

Eu sou Yhann

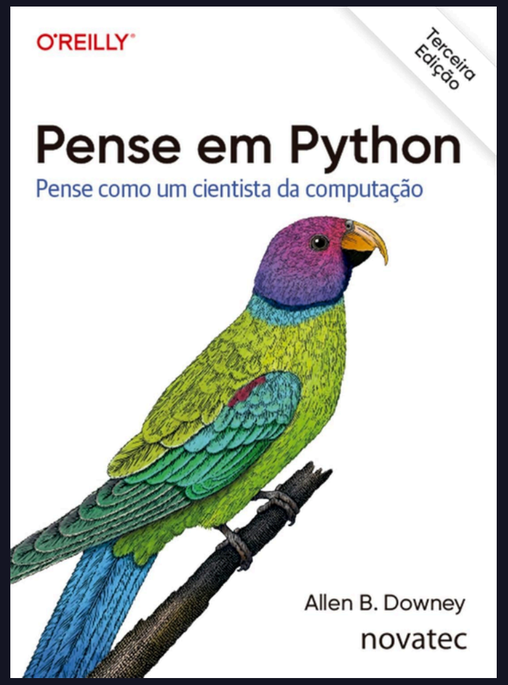
Desenvolvedor Back-end e desktop
Fundador e idealizador da Porãygua
Estudante de Ciência da computação

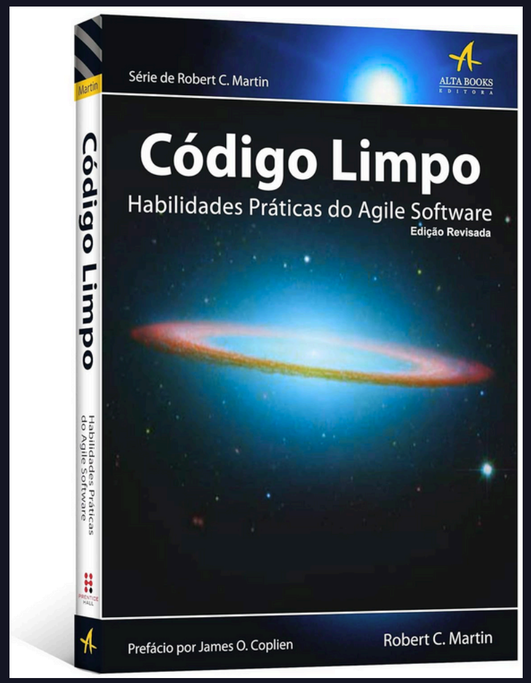
Gerente dos projetos:

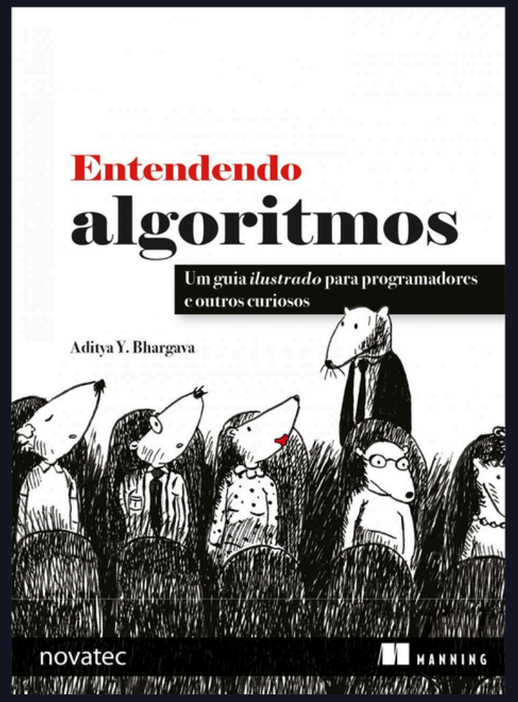

Tecnologias



Literaturas recomendadas







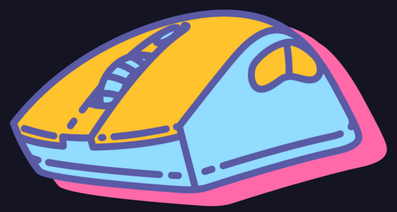
Aula 2.py

01 {

[Funções]

Conceitos, parâmetros, retornos, variações e recursividade.

}



Aula 2.py

O que são Funções {

No Python (e em quase todas as linguagens de programação), funções são blocos de código nomeados, reutilizáveis e independentes, criados para realizar uma tarefa específica. Elas representam um dos pilares da programação estruturada e são fundamentais para tornar os programas mais organizados, legíveis, reutilizáveis e manuteníveis.

}

Aula 2.py

Anatomia de uma Função {

palavra reservada python, informa que a sequencia é uma função

Nome da função

Variável(s) interna da função

```
def nome_da_funcao (parametro):
    return
```

Palavra reservada que retorna ao código um valor especifico

}

Aula 2.py

Parâmetros {

Parâmetro é uma variável local, que recebe um valor (argumento) e usa-o para executar uma função.

Um parâmetro pode receber qualquer tipo de dado inclusive Estruturas mais complexas do que os tipos primitivos, como listas, túplas, dicionários, coleções e classes.

}

Aula 2.py

Retorno

{

O retorno de uma função ocorre pela palavra reservada “return”, a qual é define o tipo da função a partir do tipo de dado que é inserido após a palavra na mesma linha.

}

Aula 2.py

Como usar uma Função{

```python
numero1 = int(input("insira seu numero: "))
numero2 = int(input("insira seu numero: "))


def somar_numeros( num1 , num2 ):
    resultado = num1 + num2
    return resultado


print(somar_numeros(numero1,numero2))
```

}

Aula 2.py

Funções Lambida λ {

Uma função lambda em Python é uma forma concisa e anônima de declarar funções.
Ela é usada quando você precisa de uma função curta, geralmente de uma única linha, sem a necessidade de nomeá-la.

```
lambda argumentos: expressão
```

}

Aula 2.py

Como usar uma função ʎ {

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

pares = list( filter( lambda x : x % 2 == 0, numeros ))

print(pares)

>> [2, 4, 6, 8, 10]
```

}

Aula 2.py

Recursão {

A recursão é um conceito fundamental da programação em que uma função chama a si mesma para resolver um problema. Esse processo se baseia em dividir tarefas complexas em subproblemas menores e mais simples. Para funcionar corretamente, toda função recursiva precisa de um caso base, que encerra as chamadas sucessivas. Embora poderosa e elegante, a recursão pode levar a erros como o estouro da pilha se mal utilizada. Seu uso é comum em algoritmos como cálculo de fatorial, sequência de Fibonacci e percursos em estruturas como árvores.

}

Aula 2.py

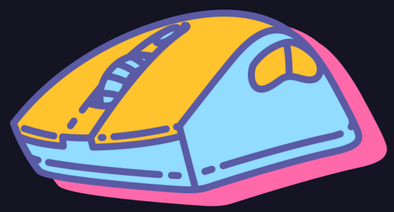
1
2
3
4
5
6
7
8
9
10
11
12
13
14

02 {

[Estruturas de dados]

Conceito, exemplos e o praticas.

}



Aula 2.py

O que são Estrutura de dados {

Estruturas de dados são formas de organizar, armazenar e gerenciar dados em um sistema de computador para otimizar o acesso e a manipulação desses dados. Elas funcionam como "recipientes" que armazenam informações de maneira sistemática, permitindo que operações como inserção, exclusão, busca e ordenação sejam realizadas de forma eficiente

No python essas Estruturas são listas, tuplas e dicionários

}

Aula 2.py

Listas: criação e características {

Assim como uma string, uma lista é uma sequencia mutável de valores quaisquer, onde cada iten da lista funciona como uma variável independente.

```
nome_da_lista = []

lista_de_numeros = [ 1 , 2 , 3 , 4]
lista_de_nomes = [“porã” , “belém”]
lista = [ 3 , “belém” , 3 , [] ]
```

- Mutabilidade
- Ordenabilidade
- Indexação
- Heterogeneidade
- Duplicidade
- Aninhamento
- Iterável

}

Aula 2.py

Listas: acesso de valores {

```
lista = [“charmander” , “bubasauro” , “squirtle”]


print(f“seu pokemon é: {lista[0]}”)
print(f“seu pokemon é: {lista[2]}”)

>> seu pokemon é: charmander
>> seu pokemon é: squirtle

}
```

Aula 2.py

Listas: Métodos auxiliares

Metodo	Descrição
.append(iten)	adiciona o item ao final da lista
.incert(iten, posição)	insere o item na posição informada
.extend(iteravel)	adiciona uma lista inteira à outra
.pop([posição])	remove o item na posição inserida
.remove(item)	remove a primeira ocorrencia do item na lista
.clear()	remove todos os itens da lista
.index(iten)	retorna o índice da primeira ocorrência do item
.copy()	copia completamente uma lista
.sort()	retorna uma lista organizada da lista original
.reverse()	retorna uma lista inversa da original
.count(iten)	retorna o número de ocorrências de um item

Aula 2.py

Tuplas: criação e características {

Uma tupla é uma coleção ordenada e imutável de elementos, ideal para representar grupos de dados que não devem ser alterados.

```
nome_da_tupla = ()

tupla_de_numeros = ( 1 , 2 , 3 , 4)
tupla_de_nomes = (“porã”, “belém”)
tupla = ( 3, “belém”, 3 , [] )
```

- Ordenabilidade
- Indexável
- Heterogeneidade
- Duplicidade
- Aninhamento
- Iterável

}

Aula 2.py

Tuplas: Metodos auxiliares {

Metodo	Descrição
.index(iten)	retorna o índice da primeira ocorrência do item
.count(iten)	retorna o número de ocorrências de um item

Aula 2.py

Dicionários: criação e características {

Um dicionário é uma estrutura de dados que armazena pares de chave-valor, assim como um dicionário real (palavra = definição).

```
meu_dicionário = {
    “nome”     : “YHANN”
    “idade”    : 15
    “dinheiro”: 2.25
    “ativo”    : true
}
}
```

- Relação chave-valor
- Chaves únicas e imutáveis
- Mutabilidade
- Iterável
- Aninhável
- Semelhanças com JSON

Aula 2.py

Dicionários: Métodos auxiliares

Metodo	Descrição
.pop(chave)	remove o item na chave correspondente
.clear()	remove todos os itens da lista
.copy()	copia completamente uma lista
.values()	retorna os valores do dicionário
.keys()	retorna as chaves do dicionário
.itens()	retorna a chave e o valor como uma tupla
.get(chave, [def])	retorna o valor da chave, se não, o valor da função
.update(dicionário)	atualiza o dicionário principal com outro dicionário

Aula 2.py

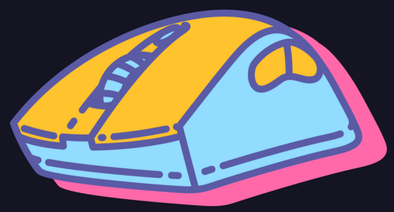
1
2
3
4
5
6
7
8
9
10
11
12
13
14

03 {

[Loops e repetidores]

For e While

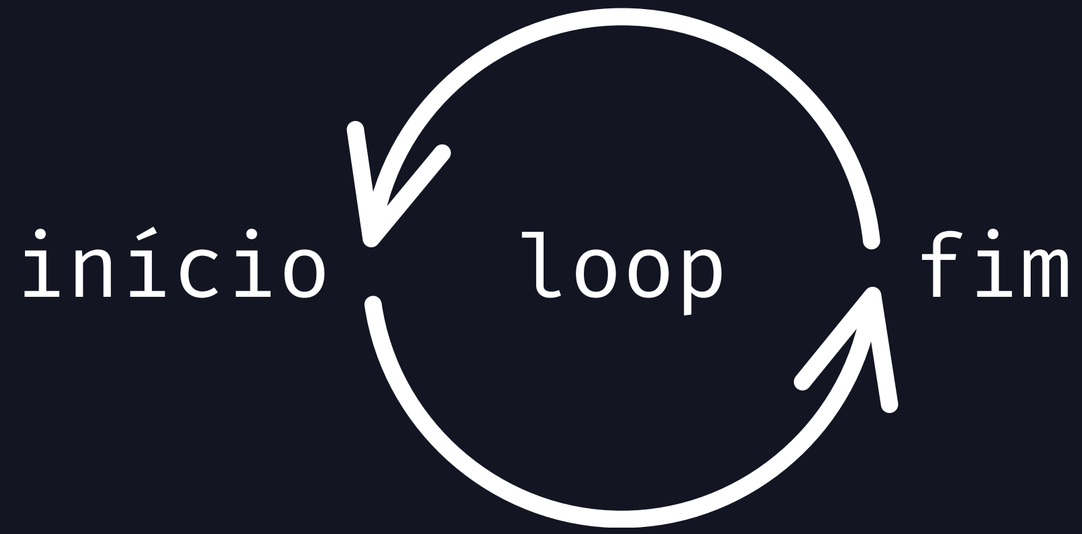
}



Aula 2.py

O que é uma Estrutura de repetição? {

Loops são estruturas de controle que permitem executar um bloco de código repetidamente, até que uma condição seja satisfeita ou até que uma sequência de elementos seja totalmente percorrida.



}

Aula 2.py

While {

O while é uma estrutura de repetição que executa um bloco de código enquanto uma condição for verdadeira. Ou seja, ele verifica a condição antes de cada iteração e só continua se ela ainda for verdadeira.

```
contador = 0

while contador < 5:
    print(contador)
    contador += 1
```

}

Aula 2.py

in e not in {

in e not in são verificadores que informam se um dado está presente em um iterável ou não. Ambos retornam um valor boleano, ou seja, verdadeiro ou falso (true ou false)

```
nomes = [“porã”, “belém”]

print(“porã” in nomes)
print(2 in nomes)

>> “true”
>> “false”
```

}

Aula 2.py

for {

O for é um loop usado para iterar sobre qualquer objeto que seja iterável (listas, strings, dicionários, tuplas, conjuntos, ranges etc.). Ele percorre os elementos um por um, na ordem em que aparecem, sem precisar de contador manual.

```
nomes = ["Ana", "Bruno", "Carlos"]

for nome in nomes:
    print(nome)
```

}