Funções – Continuação

Breve revisão sobre funções

Função - Sintaxe

Para definir funções utilizamos a seguinte sintaxe:

```
def nomeDaFuncao(parametro1, parametro2, ...):
    comando1
    comando2
    ...
    return valorRetornado
```

Chamamos a função da seguinte forma:

```
valor = nomeDaFuncao(argumento1, argumento2, ...)
```

Função - Sintaxe

Exemplo:

```
def somaValores(x, y):
    soma = x + y
    return soma


soma = somaValores(5, 6)
print(soma)
```

Função

- Importante! Uma função não é executada na sua definição, apenas quando ela é chamada!

Importante!!!

Colchetes possui um significado completamente diferente de parênteses

valores[i]

Acessa o elemento **i** da lista **valores**

valores(i)

Chama a função **valores** passando a variável **i** como argumento

Cuidado com nomes de funções

Variáveis não podem ter o mesmo nome que funções definidas no programa. Caso isso ocorra, a função será substituída pela variável

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

maiorValor = maiorValor(3, 7)
print(maiorValor)
maiorValor = maiorValor(8, 13)
print(maiorValor)
```

Chama a função **maiorValor**, associa o resultado à variável **maiorValor** e imprime. Esse trecho executa corretamente

Trata a variável **maiorValor** como função, o que está incorreto. Essa linha dará erro.

Escopo de variáveis

O termo *escopo de variáveis* está relacionado com o local do código no qual uma determinada variável pode ser acessada

Escopo de variáveis

O termo *escopo de variáveis* está relacionado com o local do código no qual uma determinada variável pode ser acessada

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
print(resultado)
```

Variáveis **valor1**, **valor2** e **maior** estão no escopo da função **maiorValor**. Portanto, não podem ser acessadas pelo programa principal

Variáveis **x**, **y** e **resultado** estão no escopo do programa principal. Portanto elas são chamadas de variáveis globais, pois podem ser acessadas por qualquer função

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    x = 10
    print("x dentro da função: {}".format(x))


x = 3
minhaFuncao()
print("x fora da função: {}".format(x))
```

O programa imprime:

```
x dentro da função: 10
x fora da função: 3
```

Quando definimos um valor de **x** dentro da função, estamos criando uma nova variável que pertence somente à função
Portanto, o programa acima possui duas variáveis chamadas **x**: a variável **x** do programa principal e a variável **x** da função **minhaFuncao**.

Funções – Retornando mais de um valor

- Uma função pode retornar mais de um valor
- Para isso, basta fazermos:

```
def minhaFuncao(lista):
    ...
    return valor1, valor2, valor3

l = [6, 7, 2, 8, 10, 3]
v1, v2, v3 = minhaFuncao(l)
```

Funções – Retornando mais de um valor

- Exemplo de retorno de mais de um valor. Uma função que retorna o menor valor da lista e o respectivo índice

```
def menorValor(lista):
    menor = lista[0]
    indiceMenor = 0
    for i in range(0, len(lista)):
        if lista[i]<menor:
            menor = lista[i]
            indiceMenor = i
    return menor, indiceMenor

l = [5, 2, 7, 4, 9, 1, 5, 7, 10]
menor, indice = menorValor(l)

print("Menor valor: {}".format(menor))
print("Posição do menor valor: {}".format(indice))
```

Funções – Argumentos opcionais

Argumentos opcionais

- Funções podem ter um ou mais parâmetros possuindo um valor padrão
- Nesse caso, não é necessário passar o valor ao chamarmos a função.

Vamos ver um exemplo:

Argumentos opcionais

- Funções podem ter um ou mais parâmetros possuindo um valor padrão
- Nesse caso, não é necessário passar o valor ao chamarmos a função.

Vamos ver um exemplo:

Tanto o valor da base quanto o valor do expoente devem ser passados à função:

```
def potencia(base, expoente):
    return base**expoente

res = potencia(4, 3)
```

Argumentos opcionais

- Funções podem ter um ou mais parâmetros possuindo um valor padrão
- Nesse caso, não é necessário passar o valor ao chamarmos a função.

Vamos ver um exemplo:

Tanto o valor da base quanto o valor do expoente devem ser passados à função:

```
def potencia(base, expoente):
    return base**expoente

res = potencia(4, 3)
```

O valor do expoente é assumido como sendo 2. Se não passarmos nenhum valor de expoente, o número é elevado ao quadrado:

```
def potencia(base, expoente=2):
    return base**expoente

res = potencia(4)
```

Argumentos opcionais

Ambos os argumentos de uma função podem ser opcionais. Nesse caso, a função pode ser chamada sem nenhum argumento:

```
def potencia(base=2, expoente=2):
    return base**expoente

res = potencia()
```

Uma função com argumentos opcionais ainda pode receber outros valores:

```
def potencia(base=2, expoente=2):
    return base**expoente

res = potencia(3, 3)
```

Argumentos opcionais

Importante: parâmetros opcionais devem sempre estar após parâmetros não opcionais

Argumentos opcionais

Importante: parâmetros opcionais devem sempre estar após parâmetros não opcionais

Código inválido, **expoente** deve possuir um valor padrão, ou aparecer antes do parâmetro **base**:

```
def potencia(base=2, expoente):
    return base**expoente
```

Passagem de argumento por nome

Passagem de argumento por nome

Também é possível passarmos valores à uma função utilizando o nome do parâmetro:

```
def potencia(base=2, expoente=2):
    return base**expoente

res = potencia(base=3, expoente=4)
```

Nesse caso, podemos passar os valores em qualquer ordem:

```
def potencia(base=2, expoente=2):
    return base**expoente

res = potencia(expoente=4, base=3)
```

Passagem de argumento por nome

- Argumentos opcionais e passagem por nome são úteis quando uma função possui muitos argumentos;

Passagem de argumento por nome

- Argumentos opcionais e passagem por nome são úteis quando uma função possui muitos argumentos;
- Por exemplo, suponha que criamos um função que plota dados na tela. Além dos dados em si, a função precisa receber algumas propriedades visuais do gráfico (cor, tipo de linha, etc);

Passagem de argumento por nome

- Argumentos opcionais e passagem por nome são úteis quando uma função possui muitos argumentos;
- Por exemplo, suponha que criamos um função que plota dados na tela. Além dos dados em si, a função precisa receber algumas propriedades visuais do gráfico (cor, tipo de linha, etc);
- Exemplo de definição da função:

```
def plotaGrafico(x, y, tipo="linha", corLinha="azul", corPontos="preto",
                 corFundo="branco", nomeEixoX="Tempo", nomeEixoY="Amplitude"):
    ...
```

Passagem de argumento por nome

- Argumentos opcionais e passagem por nome são úteis quando uma função possui muitos argumentos;
- Por exemplo, suponha que criamos um função que plota dados na tela. Além dos dados em si, a função precisa receber algumas propriedades visuais do gráfico (cor, tipo de linha, etc);
- Exemplo de definição da função:

```
def plotaGrafico(x, y, tipo="linha", corLinha="azul", corPontos="preto",
                 corFundo="branco", nomeEixoX="Tempo", nomeEixoY="Amplitude"):
    ...
```

- Essa função poderia ser chamada de diversas formas, dependendo do que se deseja alterar no gráfico:

```
plotaGrafico(x, y)
plotaGrafico(x, y, corLinha="vermelho")
plotaGrafico(x, y, nomeEixoX="Posicao", nomeEixoY="Velocidade")
```

Exercício

Faça uma função que recebe como entrada os seguintes parâmetros:

1. Dois números **n1** e **n2**
2. Uma variável booleana **deveMultiplicar**

O valor retornado pela função depende do valor da variável **deveMultiplicar**:

- Se **deveMultiplicar**=False, a função retorna a soma dos números **n1** e **n2**
- Se **deveMultiplicar**=True, a função retorna o produto dos números **n1** e **n2**

Se apenas os valores **n1** e **n2** forem passados à função, você deve assumir que o usuário quer somar os dois números.

Exercício

Faça uma função que recebe como entrada os seguintes parâmetros:

1. Dois números **n1** e **n2**
2. Uma variável booleana **deveMultiplicar**

O valor retornado pela função depende do valor da variável **deveMultiplicar**:

- Se **deveMultiplicar**=False, a função retorna a soma dos números **n1** e **n2**
- Se **deveMultiplicar**=True, a função retorna o produto dos números **n1** e **n2**

Se apenas os valores **n1** e **n2** forem passados à função, você deve assumir que o usuário quer somar os dois números.

```
def somaMult(n1, n2, deveMultiplicar=False):
    if deveMultiplicar==False:
        return n1+n2
    else:
        return n1*n2
```

Exercício

Faça um código possuindo uma função, chamada **media**, que calcula a média de uma lista. A função recebe a lista e retorna a media dos valores.

Exercício - Solução

Faça um código possuindo uma função, chamada **media**, que calcula a média de uma lista. A função recebe a lista e retorna a media dos valores.

```
def media(lista):
    soma = 0
    for i in range(len(lista)):
        soma = soma + lista[i]
    valorMedio = soma/len(lista)
    return valorMedio
```

Exercício

Faça um código possuindo uma função, chamada **variancia**, que calcula a variância dos valores em uma lista.

Fórmula da variância:

$$V = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)^2$$

N: Número de valores (tamanho da lista)
x_i: i-ésimo valor
μ_x: Média dos valores

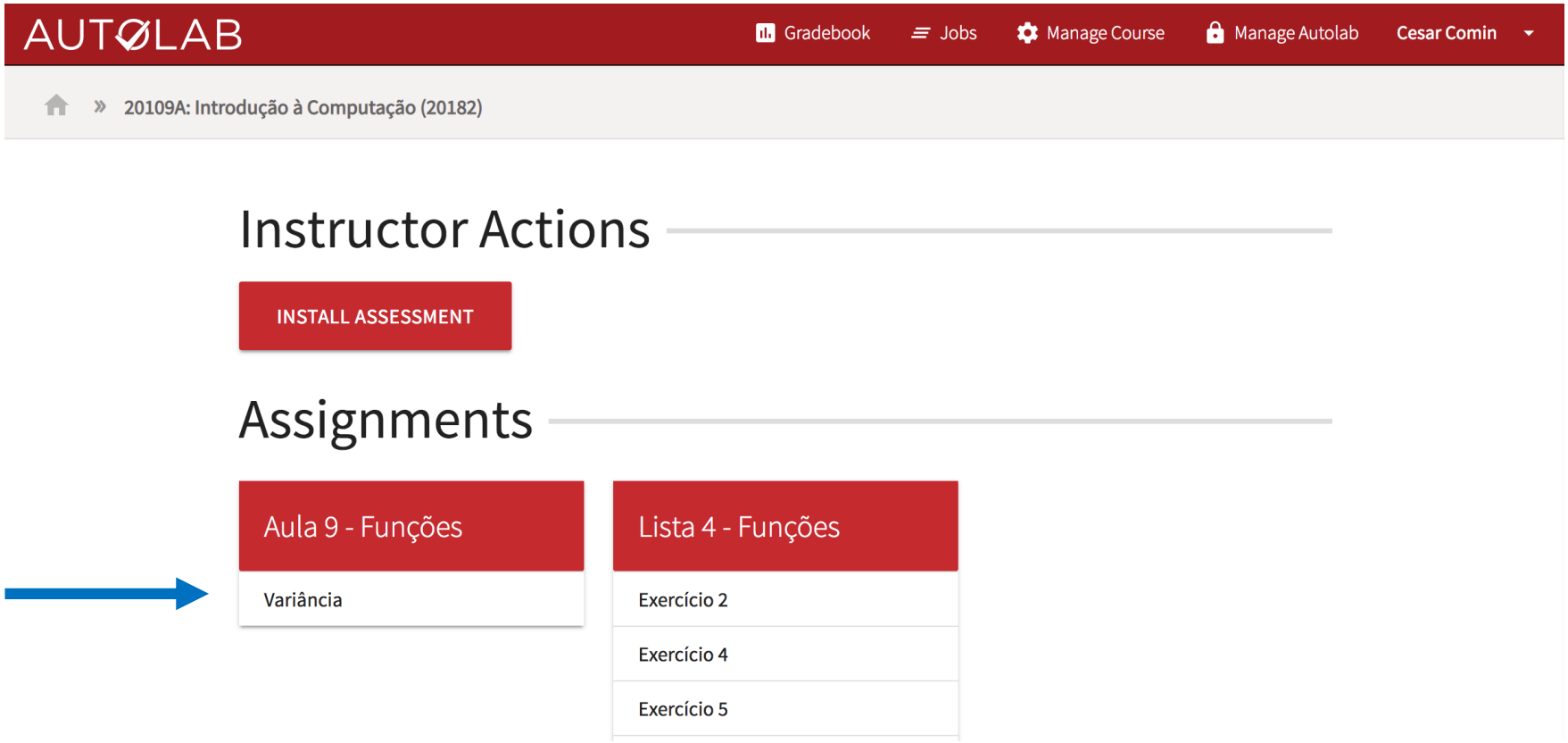
A variância é uma propriedade de um conjunto de valores que indica o grau de espalhamento dos valores em torno da média

Por exemplo, no caso da lista [2, 3, 7], a média dos valores é 4, e portanto a variância é calculada como

$$V = \frac{1}{3}[(2-4)^2 + (3-4)^2 + (7-4)^2]$$

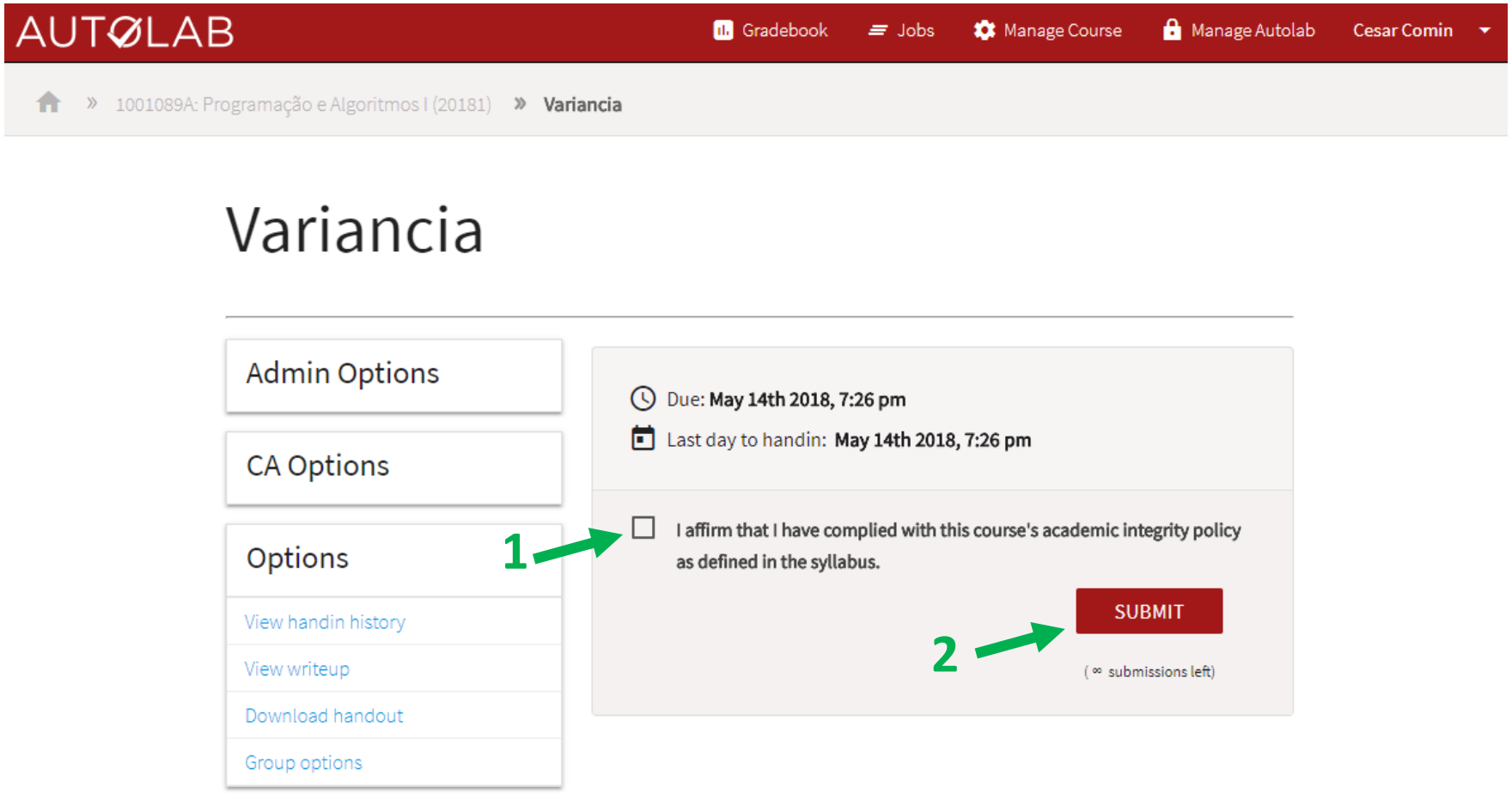
Autolab

Verifique o código gerado utilizando o site ***autolab.ufscar.br***



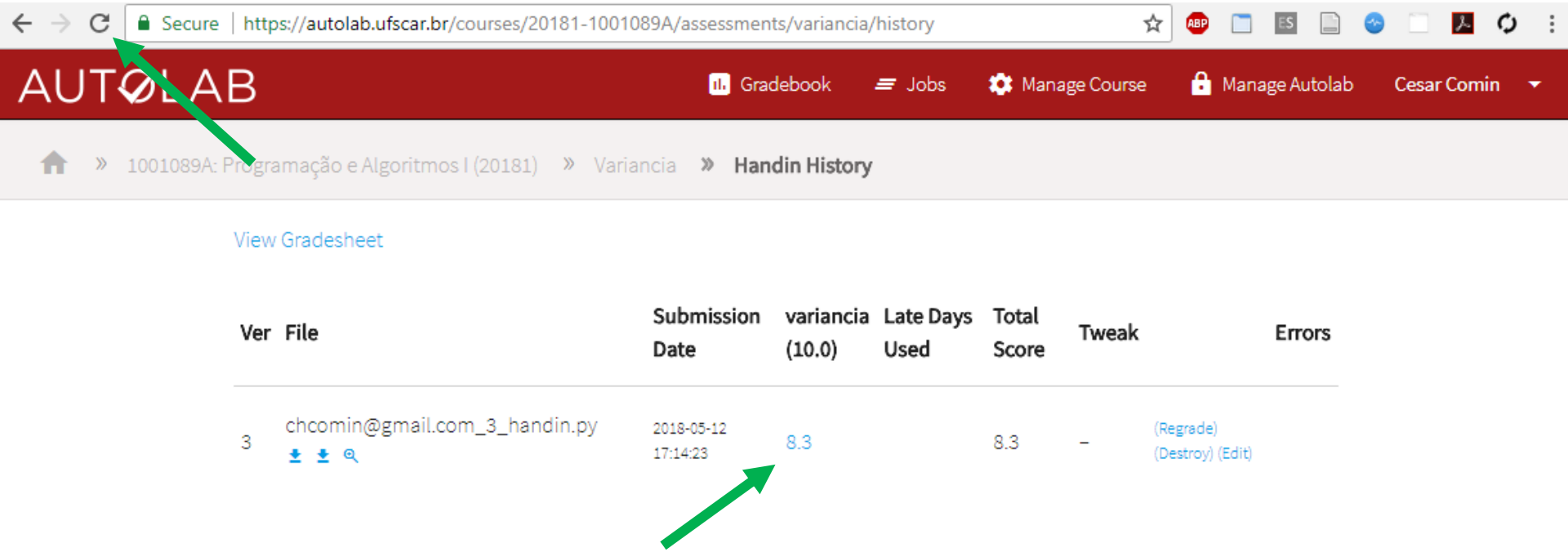
Autolab

Submeta o código produzido clicando na caixinha 1 e depois no botão submit



Autolab

Após submeter a solução, **espere alguns segundos**, recarrege a página e clique na nota obtida



Autolab

O sistema mostrará para você os casos de teste que deram errado

AUTØLAB Gradebook Jobs Manage Course Manage Autolab Cesar Comin

» 1001089A: Programação e Algoritmos I (20181) » Variancia

Feedback for Variancia - variancia (chcomin@gmail.com)

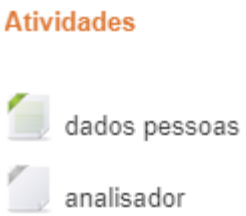
```
Autograder [Sat May 12 17:14:25 2018]: Received job 20181-1001089A_variancia_3_chcomin@gmail.com:198
Autograder [Sat May 12 17:14:30 2018]: Success: Autodriver returned normally
Autograder [Sat May 12 17:14:30 2018]: Here is the output from the autograder:
---
Autodriver: Job exited with status 0

Resultado: 8.203125
Testando sua solucao...
Teste 1 deu errado!
O resultado para [2, 7, 6, 4, 8, 1, 10, 4] deveria ser 8.187500 mas deu 8.203125
Teste 2: correto!
Teste 3: correto!
Teste 4: correto!
Teste 5: correto!
Teste 6: correto!
{"scores": {"variancia": 8.333333333333334}}

Score for this problem: 8.3
```

Exercício variância - Solução

```
def media(lista):
        soma = 0
        for i in range(len(lista)):
                soma = soma + lista[i]
        media = soma/len(lista)

        return media

def variancia(lista):
        valorMedio = media(lista)
        soma = 0
        for i in range(len(lista)):
                soma = soma + (lista[i]-valorMedio)**2
        var = soma/len(lista)

        return var
```

Análise de dados

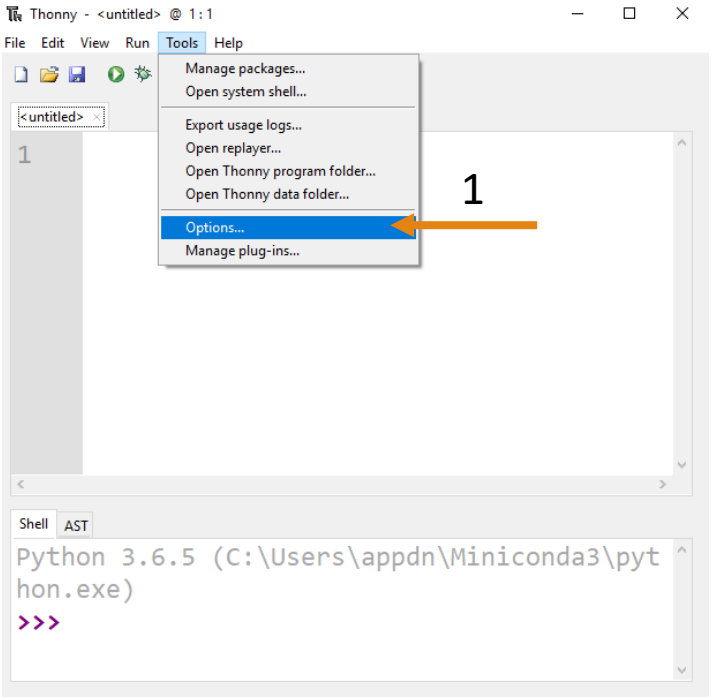
- Iremos analisar um conjunto de dados contendo informações sobre o nome, idade, altura e salário de um conjunto de pessoas
- Para isso, utilizaremos os dados disponíveis no AVA e o módulo **analisador**, também disponível no AVA

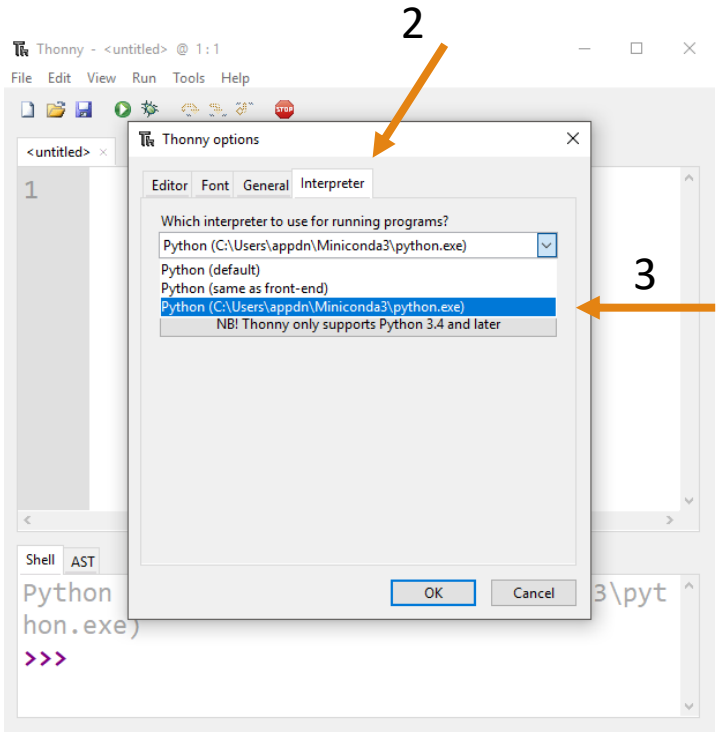


- Baixe os arquivos “dados_pessoas.txt” e “analisador.py” e coloque-os na mesma pasta onde você irá criar os seus códigos

Análise de dados

- Ajuste o Thonny para utilizar outra versão do Python, que possui a biblioteca de plotagem (matplotlib) já instalada.





Análise de dados

20 primeiras linhas do arquivo dados_pessoas.txt :

```
 1 Nome Idade Altura Salario
 2 Rita 35 1.72 3721.4
 3 Andy 34 1.71 1429.4
 4 Donald 33 1.82 987.0
 5 Mamie 31 1.79 2266.8
 6 Selma 27 1.82 2136.6
 7 Sharron 12 1.66 679.2
 8 Scott 31 1.80 0.0
 9 Karen 35 1.83 3453.5
10 Kevin 39 1.89 3391.2
11 Elizabeth 32 1.81 2911.1
12 Sarah 35 1.79 3313.2
13 Rochelle 29 1.70 225.3
14 Raul 39 1.67 5272.1
15 Edward 44 1.65 4343.8
16 John 27 1.80 0.0
17 Mary 28 1.85 0.0
18 Lucas 28 1.78 1607.2
19 Diana 51 1.54 7631.3
20 Bonnie 34 1.77 277.2
```

Análise de dados

Para ler os dados, faça

```
import analisador

nomes, idades, alturas, salarios = analisador.leDados('dados_pessoas.txt')
```

Análise de dados

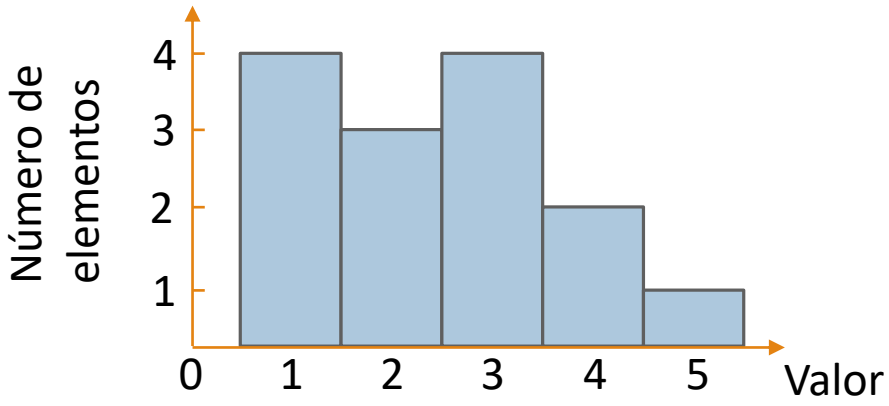
Para conhecer um pouco melhor os dados, vamos primeiro plotar histogramas de cada variável

Análise de dados

Para conhecer um pouco melhor os dados, vamos primeiro plotar histogramas de cada variável

Um histograma é uma quantificação do número de elementos em uma lista possuindo cada valor único da lista

3	1	1	2	3	4	5	2	1	3	4	2	3	1



Análise de dados

Para plotar histogramas dos dados, faça

```
import analisador

nomes, idades, alturas, salarios = analisador.leDados('dados_pessoas.txt')

analisador.histograma(idades, nomeEixoX='Idade')
analisador.histograma(alturas, nomeEixoX='Alturas')
analisador.histograma(salarios, nomeEixoX='Salarios')
```

Exercício

Considere os dados das pessoas. Faça um programa que encontra o nome da pessoa mais velha

* O seu código deve começar com os comandos:

```
import analisador

nomes, idades, alturas, salarios = analisador.leDados('dados_pessoas.txt')
```

Exercício

Considere os dados das pessoas. Faça um programa que encontra o nome da pessoa mais velha

```
import analisador

nomes, idades, alturas, salarios = analisador.leDados('dados_pessoas.txt')

maior = idades[0]
indiceMaior = 0
for i in range(0, len(idades)):
    if idades[i]>maior:
        maior = idades[i]
        indiceMaior = i
nomeMaisVelho = nomes[indiceMaior]

print('A pessoa mais velha é {}'.format(nomeMaisVelho))
print('Essa pessoa possui {} anos'.format(maior))
```

Exercício

Considere os dados das pessoas. Faça um programa que encontra o nome da pessoa mais velha

Como podemos reescrever esse programa de forma a deixar ele mais bem estruturado?

Exercício

Considere os dados das pessoas. Faça um programa que encontra o nome da pessoa mais velha

Como podemos reescrever esse programa de forma a deixar ele mais bem estruturado?

Podemos definir uma função para executar a tarefa de encontrar o nome e idade da pessoa mais velha

Exercício

```
import analisador

def maiorIdade(nomes, idades):
    maior = idades[0]
    indiceMaior = 0
    for i in range(0, len(idades)):
        if idades[i]>maior:
            maior = idades[i]
            indiceMaior = i
    nomeMaisVelho = nomes[indiceMaior]

    return nomeMaisVelho, maior

nomes, idades, alturas, salarios = analisador.leDados('dados_pessoas.txt')

nome, idade = maiorIdade(nomes, idades)

print('A pessoa mais velha é {}'.format(nome))
print('Essa pessoa possui {} anos'.format(idade))
```

Exercício

```
import analisador

def maiorIdade(nomes, idades):
    maior = idades[0]
    indiceMaior = 0
    for i in range(0, len(idades)):
        if idades[i]>maior:
            maior = idades[i]
            indiceMaior = i
    nomeMaisVelho = nomes[indiceMaior]

    return nomeMaisVelho, maior

nomes, idades, alturas, salarios = analisador.leDados('dados_pessoas.txt')

nome, idade = maiorIdade(nomes, idades)

print('A pessoa mais velha é {}'.format(nome))
print('Essa pessoa possui {} anos'.format(idade))
```

Note que a função **maiorIdade** poderia ser ainda mais genérica. Ela pode receber uma lista qualquer de dados e outra lista numérica, e retornar o maior valor da lista numérica e o respectivo valor da lista de dados

Exercício

Faça uma função que recebe como entrada dois valores de idade, e retorna uma lista possuindo a altura de todas as pessoas dentro desse intervalo de idade.

Por exemplo, se a função receber os valores 23 e 29, ela retorna uma lista com as alturas de todas as pessoas possuindo idades entre 23 e 29.

* Dica:

```
alturasNaFaixa = []
for i in range(0, len(idades)):
    if idades[i]>menorIdade and idades[i]<maiorIdade:
```

Exercício

```
import analisador

def alturasFaixaIdade(idades, alturas, menorIdade, maiorIdade):
    alturasNaFaixa = []
    for i in range(0, len(idades)):
        if idades[i]>menorIdade and idades[i]<maiorIdade:
            alturasNaFaixa.append(alturas[i])

    return alturasNaFaixa

nomes, idades, alturas, salarios = analisador.leDados('dados_pessoas.txt')

alturasIntervalo = alturasFaixaIdade(idades, alturas, 30, 40)
```