PYTHON

FUNÇÕES

Python e ciência dos dados

COMPONENTES EM PYTHON

- Tipicamente, programs em Python são escritos combinando funções e classes escritas pelo programador com módulos já existentes.
- Componentes de programa em Python podem ser:
- funções
- classes,
- modulos
  - Um módulo é um arquivo que contem deifnições de funções e classes.
- packages.
  - *Módulos podem ser agrupadso para formar uma coleção, ou package*, para uma tarefa específica (gerar gráficos, processamento de texto)

- Por que criar um programa a partir de componentes?
  - Dividir e conquistar
  - *reutilização de software*
  - *evitar a repetição de código*

FUNÇÕES

- Funções permitem a modularização de programas.
  - Toda variável criada dentro de uma função é local – conhecida no corpo da função.

- Para usar funções que são definidas em um módulo, o programa deve importar o módulo, usando a palavra-chave ***import***

- Uma vez importado, invoca-se uma função, usando o ***nomeDoModulo.nomeDaFuncao()***

```
import math

print(math.sqrt(4))
```

O MÓDULO DE FUNÇÕES MATH

- O módulo **math** contem funções para alguns cálculos matemáticos.

Métodos	Descrição	Método	Descrição
acos(*x*)	arco coseno (radiano)	**fmod(*x*, *y*)**	módulo como um float
asin(*x*)	arco seno	**hypot(*x*, *y*)**	hipotenusa de um triangulo com lados x e y
atan(*x*)	arto tangente	**log(*x*)**	logaritmo natural
ceil(*x*)	arrendonda x para o próximo inteiro maior que x	**log10(*x*)**	logaritmo de x na base 10
cos(*x*)	coseno (radiano)	**pow(*x*, *y*)**	x^y
exp(*x*)	e^x	**sin(*x*)**	seno (radiano)
fabs(*x*)	valor absoluto de x	**sqrt(*x*)**	raiz quadrada
floor(*x*)	arrendonda x para o próximo inteiro menor que x	**tan(*x*)**	tangente

ACESSANDO INFORMAÇÕES DE MÓDULOS

- Acessando identificadores presentes em um módulo:
  - No interpretador digite: dir(<modulo>)
    - Se o módulo não for nativo do Python, importe-o antes.
    - ex.

    import math

    dir(math)
- Use o **help** para saber a funcionalidade de cada identificador
  - help(math.cos)

```
>>> help(math.cos)
Help on built-in function cos in module math:

cos(...)
    cos(x)

    Return the cosine of x (measured in radians).
```

USANDO O PREENCHIMENTO DA GUIA IPYTHON PARA DESCOBERTA

- Pode-se visualizar a documentação de um módulo no IPython digitando uma parte de um identificador e pressionando <*Tab*>
  - o IPython completa o identificador para você ou fornece uma lista de identificadores que começam com o que você digitou até agora.

```
In [1]: import math

In [2]: ma<Tab>
           map          %macro       %%markdown
           math         %magic       %matplotlib
           max()        %man
```

USANDO O PREENCHIMENTO DA GUIA IPYTHON PARA DESCOBERTA

- Para visualizar uma lista de identificadores definidos em um módulo, digite o nome do módulo seguido de ponto (.), em seguida, pressione Tab:

```
In [3]: math.<Tab>
        acos()      atan()      copysign()  e           expm1()
        acosh()     atan2()     cos()       erf()       fabs()
        asin()      atanh()     cosh()      erfc()      factorial() >
        asinh()     ceil()      degrees()   exp()       floor()
```

- Os identificadores seguidos por parênteses são funções (ou métodos).
- Identificadores de uma única palavra que começam com uma letra maiúscula representam nomes de classes
- Identificadores em minúsculas sem parênteses, como pi e *e*, são variáveis.

PYTHON STANDARD LIBRARY

- Programas em Python são escritos combinando funções e classes que você cria com funções e classes preexistentes definidas em módulos, como aqueles na biblioteca padrão do Python e outras bibliotecas.
  - Um dos principais objetivos da programação é evitar “reinventar a roda”.

- A biblioteca padrão Python é fornecida com a linguagem Python. Seus pacotes e módulos contêm recursos para uma ampla variedade de tarefas de programação
  - Você pode ver uma lista completa dos módulos de biblioteca padrão em:

https://docs.python.org/3/library/

ALGUNS MÓDULOS POPULARES

`collections`—Data structures beyond lists, tuples, dictionaries and sets.

Cryptography modules—Encrypting data for secure transmission.

`csv`—Processing comma-separated value files (like those in Excel).

`datetime`—Date and time manipulations. Also modules `time` and `calendar`.

`decimal`—Fixed-point and floating-point arithmetic, including monetary calculations.

`doctest`—Embed validation tests and expected results in docstrings for simple unit testing.

`gettext` and `locale`—Internationalization and localization modules.

`json`—JavaScript Object Notation (JSON) processing used with web services and NoSQL document databases.

`math`—Common math constants and operations.

`os`—Interacting with the operating system.

`profile`, `pstats`, `timeit`—Performance analysis.

`random`—Pseudorandom numbers.

`re`—Regular expressions for pattern matching.

`sqlite3`—SQLite relational database access.

`statistics`—Mathematical statistics functions such as `mean`, `median`, `mode` and `variance`.

`string`—String processing.

`sys`—Command-line argument processing; standard input, standard output and standard error streams.

`tkinter`—Graphical user interfaces (GUIs) and canvas-based graphics.

`turtle`—Turtle graphics.

`webbrowser`—For conveniently displaying web pages in Python apps.

DEFINIÇÃO DE FUNÇÕES

- Funções devem ser definidas antes de seu uso.
- Em Python, defini-se funções como:

def *nomeDafunção(listaDeParametros):*

sentenças

```
def square( y ):
   return y * y

for x in range( 1, 11 ):
   print(square( x ))
```

DEFINIÇÃO DE FUNÇÕES

```
def square(y):
...: """Calcula o quadrado de y."""
...: return y ** 2
```

- Para ver o ***docstring*** de uma função digite o nome da função seguido por um ponto de interrogação (?):

Se o código-fonte da função for acessível a partir do IPython, você pode usar ?? para exibir o código-fonte da função:

```
In [3]: square?
Signature: square(y)
Docstring: Calcula o quadrado de y
File:      c:\users\joão\<ipython-input-37-61ae056c0215>
Type:      function
```

FUNÇÕES COM MÚLTIPLOS ARGUMENTOS

- Vamos definir uma função máximo que retorna o maior dos três valores—note que a função funciona para diferentes tipos.

```
def maximum(value1, value2, value3):
...: """Retorna o maior de tres valores."""
...: max_value = value1
...: if value2 > max_value:
...: max_value = value2
...: if value3 > max_value:
...: max_value = value3
...: return max_value
```

```
maximum(12, 27, 36)
Out[2]: 36

In [3]: maximum(12.3, 45.6, 9.7)
Out[3]: 45.6

In [4]: maximum('yellow', 'red', 'orange')
Out[4]: 'yellow'
```

GERAÇÃO DE NÚMEROS ALEATÓRIOS

- A geração de números aleatórios está relacionado a varias aplicações em computação
  - desde a simulação de probabilidade a aplicação em jogos.

- Em Python números aleatórios podem ser gerados por meio da função r***andom.randrange()*** do módulo ***random***.
  - ***random.randrange(x,y) gera um número inteiro no intervalo [x, y-1]***

EXEMPLO RANDOM

```
import random

for roll in range(10):
        print(random.randrange(1, 7), end=' ')
```

4 2 5 5 4 6 4 6 1 5

random.random() gera valores no intervalo [0,1)

SIMULANDO PROBABILIDADES

```
Import random

prob = 0.4
cont = 0
for roll in range(10000):
    if random.random() < prob:
        cont+=1

print(cont)
```

Probabilidade de determinado evento

Simulação de probabilidade

Evento ocorre com *prob* de chance

RANDOM *SEED*

- A função *randrange* gera números pseudo-aleatórios, com base em um cálculo interno que começa com um valor numérico conhecido como *seed* (semente).

- Chamando *randrange* repetidamente produz uma sequência de números que parecem ser aleatórios,
  - Porque cada vez que você inicia um nova sessão interativa ou executa um script que usa as funções do módulo aleatório, Python usa internamente um valor de semente diferente.

AJUSTANDO SEED

- Quando você está depurando erros de lógica em programas que usam dados gerados aleatoriamente, pode ser útil usar a mesma sequência de números aleatórios.

```
random.seed(32)
In [5]: for roll in range(10):
...: print(random.randrange(1, 7), end=' ')
...:
1 2 2 3 6 2 4 1 6 1

In [6]: for roll in range(10):
...: print(random.randrange(1, 7), end=' ')
...:
1 3 5 3 1 5 6 4 3 5

In [7]: random.seed(32)
In [8]: for roll in range(10):
...: print(random.randrange(1, 7), end=' ')
...:
1 2 2 3 6 2 4 1 6 1
```

EXEMPLO

- Use um laço *for*, *randrange* e um expressão condicional para simular 20 lançamentos de moeda, exibindo H para cara e T para coroa, todos na mesma linha, cada um separado por um espaço.

```
import random
In [2]: for i in range(20):
...: print('H' if random.randrange(2) == 0 else 'T', end=' ')
...:
T H T T H T T T T H T H H T H T H H H H
```

EXERCÍCIO

- *CRAPS – O jogador lança dois dados. Então soma-se as faces dos dados:*
  - *Se a soma for 7 ou 11 na primeira rodada, o jogador ganha.*
  - *Se for 2, 3 ou 12 na primeira rodada (ou CRAPS) o jogador perde .*
  - *Se a soma for 4, 5, 6, 8, 9 ou 10 na primeira rodada. Então esta soma se torna o ponto do jogador.*
    - *Para ganhar, o jogador continua jogando o dado até a soma ser igual a seu ponto.*
    - *O jogador perde se a soma for igual a 7.*

RETORNANDO MÚLTIPLOS VALORES DE UMA FUNÇÃO

- As vezes é útil retornar mais de um valor, como em *roll_disse*
  - *Que pode ser feito em Python usando* uma tupla - uma sequência de valores imutável (isto é, não modificável).
- Esse recurso é conhecido como empacotamento de tupla.
  - Os parênteses são opcionais, mas recomendamos o uso para maior clareza.

```
def roll_dice():
    """Roll two dice and return their face values as a tuple."""
    die1 = random.randrange(1, 7)
    die2 = random.randrange(1, 7)
    return (die1, die2) # empacota as saída em um tupla
```

RECEBENDO MAIS DE UM VALOR DE UMA FUNÇÃO

- Para usar os valores de uma tupla, você pode atribuí-los a uma lista de variáveis separadas por vírgulas, que desempacota a tupla.
  - O número de variáveis à esquerda de '=' deve corresponder ao número de elementos no tupla; caso contrário, ocorre um **ValueError**.

```
def display_dice(dice):
    """Display one roll of the two dice."""
    die1, die2 = dice # desempacota a tupla
    print(f'Player rolled {die1} + {die2} = {sum(dice)}')
```

FORMATTED STRING

- Ainda em *display_dice*(), A letra f em print indica que é uma f-string.
  - Você especifica onde inserir valores usando marcadores de posição delimitados por chaves.
  - Expressões de texto de substituição podem conter valores, variáveis ou outras expressões, como cálculos ou chamadas de função.

```
def display_dice(dice):
    """Display one roll of the two dice."""
    die1, die2 = dice # desempacota a tupla
    print(f'Player rolled {die1} + {die2} = {sum(dice)}')
```

https://realpython.com/python-f-strings/

PRIMEIRA RODADA

- Você pode ganhar ou perder na primeiro rodada ou em qualquer rodada subsequente.
  - A variável *game_status* mantém o controle do status de vitória / derrota.

- A linha:

```
if sum_of_dice in (7, 11): # vence
```

- testa se a tupla (7, 11) contém o valor de *sum_of_dice*.
  - O operando à direita do operador **in** pode ser qualquer iterável.

EXEMPLO IPYTHON

- Empacote uma tupla de aluno com o nome 'Sue' e a lista [89, 94, 85],
- exiba a tupla e, em seguida, descompacte-a em variáveis nome e notas e exiba seus valores.

```
student = ('Sue', [89, 94, 85])

In [2]: student
Out[2]: ('Sue', [89, 94, 85])

In [3]: name, grades = student
In [4]: print(f'{name}: {grades}')

Sue: [89, 94, 85]
```

ARGUMENTOS EM QUALQUER ORDEM

- Ao chamar funções, você pode usar os identificadores dos argumentos para passar argumentos em qualquer ordem.

```
def nomeIdade(nome, idade):
...:     """imprime nome e idade"""
...:     print(f'meu nome e {nome} e idade {idade}')
...:     return
```

```
nomeIdade(idade=23,nome= 'mike')
meu nome e mike e idade 23
```

```
nomeIdade(23, 'mike')
meu nome e 23 e idade mike
```

NÚMERO ARBITRÁRIO DE ARGUMENTOS

- Funções com listas de argumentos arbitrários, como as funções integradas min e max, podem receber qualquer número de argumentos. Ex.

```
min(88, 75, 96, 55, 83)
```

- A documentação da função mostra que **min** tem dois parâmetros obrigatórios (arg1 e arg2) e um terceiro parâmetro opcional da forma *** args**, indicando que a função pode receber qualquer número de argumentos adicionais.
- O * antes do nome do parâmetro diz a Python para empacotar quaisquer argumentos restantes em uma tupla que é passada para o parâmetro args.
  - Na chamada acima, o parâmetro arg1 recebe 88, o parâmetro arg2 recebe 75 e o parâmetro args recebe a tupla (96, 55, 83).

DEFININDO FUNÇÕES COM NÚMERO ARBITRÁRIO DE ARGUMENTOS

- Vamos definir uma função média que pode receber qualquer número de argumentos:

```
def average(*args):
...: return sum(args) / len(args)
```

- Se a função tem vários parâmetros, o parâmetro * args deve ser o parâmetro mais à direita.

```
In [2]: average(5, 10)
Out[2]: 7.5

In [3]: average(5, 10, 15)
Out[3]: 10.0

In [4]: average(5, 10, 15, 20)
Out[4]: 12.5
```

O OPERADOR *

- Você pode desempacotar uma tupla, lista ou outros elementos iteráveis para passá-los como argumentos usando o operador *

```
In [5]: grades = [88, 75, 96, 55, 83]
In [6]: average(*grades)

Out[6]: 79.4
```

A chamada anterior é equivalente a

```
average(88, 75, 96, 55, 83)
```

ESCOPO DE VARIÁVEIS

- Cada identificador possui um escopo que determina onde você pode usá-lo em seu programa.

- Escopo Local - O identificador de uma variável local tem escopo local.
  - Está "no escopo" apenas a partir de sua definição até o final do bloco da função.
  - Ele “sai do escopo” quando a função retorna para seu chamador. Portanto, uma variável local pode ser usada apenas dentro da função que a define.

- Escopo global - identificadores definidos fora de qualquer função (ou classe) têm escopo global - estes podem incluir
- funções, variáveis e classes.
  - Variáveis com escopo global são conhecidas como variáveis globais.
  - Identificadores com escopo global podem ser usados em um arquivo .py ou sessão interativa em qualquer lugar após eles estão definidos.

ESCOPO

- Pode-se acessar uma variável global em uma função

```
In [1]: x = 7
In [2]: def access_global():
...: print('x printed from access_global:', x)

In [3]: access_global()
x printed from access_global: 7
```

- Mas, por padrão não é possível alterá-la.

```
In [4]: def try_to_modify_global():
...: x = 3.5
...: print('x printed from try_to_modify_global:', x)

In [5]: try_to_modify_global()
x printed from try_to_modify_global: 3.5

In [6]: x
Out[6]: 7
```

ESCOPO

- Para alterar uma variável global no corpo de uma função usa-se o comando **global**

```
In [1]: x = 7
```

```
In [7]: def modify_global():
...: global x
...: x = 'hello'
...: print('x printed from modify_global:', x)
...:

In [8]: modify_global()
x printed from modify_global: hello

In [9]: x
Out[9]: 'hello'
```

ESCOPO DE VARIÁVEIS

```
x = 1 # variável global

def a():
    x = 25  # váriavel local x, esconde a variavel global
    print( "\n x local em a é ", x, “apos entrar em a()“)
    x += 1
    print(“ x local em a é", x, “antes de sair de a()“)

# altera a variável global x
def b():
    global x      # designa escopo global a x
    print("\n x global é", x, “quando entra em b()“)
    x *= 10
    print(“x global é", x, “saindo de b()“)

print(“ x global", x)
x = 7
print(“x global", x)
```

O que acontece com x com as chamadas
a()
b()
a()
b()

ESCONDENDO FUNÇÕES

- Se uma variável for definida com o nome de uma função, ela esconde a função, tornando-o inacessível em seu código.
  - Quando usada função, ocorre um **TypeError**:

```
In [10]: sum = 10 + 5
In [11]: sum
Out[11]: 15
In [12]: sum([10, 5])
---------------------------------------------------------------------------
TypeError Traceback (most recent call last)
<ipython-input-12-1237d97a65fb> in <module>()
----> 1 sum([10, 5])
TypeError: 'int' object is not callable
```

FROM/IMPORT

- Muitas vezes um programa necessita de apenas alguns identificadores de um módulo. Nesses casos, é útil importar somente a função necessária.

- Por meio da sentença ***from/import***, é possível importar um ou mais identificadores de um módulo.
- Ex.
  - **from math import sqrt**
  - **from math import sin, cos, tan**

- Nestes casos, usa-se o identificador importado sem o operador '.'.

```
from math import sqrt

print(sqrt(4))
```

EVITE IMPORTAR TUDO!

- Você pode importar todos os identificadores definidos em um módulo, usando:

```
from modulename import *
```

- No entanto, essa prática deve ser evitada, pois pode provocar erros inesperados

```
In [4]: e = 'hello‘

In [5]: from math import *

In [6]: e
Out[6]: 2.718281828459045
```

IMPORT AS

- **import as** permite criar um nome alternativo para o identificador importado.
- **Ex.**

  import random as randomModule

- importa o módulo random, mas especifica o nome randomModule
  - Dessa forma, acessa-se o módulo pelo novo nome
  - randomModule.randint()

- Também é possível especificar nome alternativo para identificadores em um módulo.
- **Ex.**

  from math import sqrt as raizQuadrada

ARGUMENTOS PADRÃO

- Quando define-se uma função, pode-se especificar argumentos padrões e atribuir valores padrões a eles.
  - Na chamada da função, quando algum argumento padrão é omitido, o interpretador assume o valor padrão para este.
- Argumentos padrão devem aparecer à direita de qualquer argumento não padrão.

```
def volume(comprimento, largura = 1, altura = 1 ):
                return comprimento * largura * altura
```

ARGUMENTOS PADRÃO

```
def volume( comprimento = 1, largura = 1, altura = 1 ):
        return comprimento * largura * altura
```

```
volume()   # comprimento = 1, largura = 1, altura = 1
volume(10) # comprimento = 10, largura = 1, altura = 1
volume(12, 2)  # comprimento = 12, largura = 2, altura = 1
volume(2,3,4) # comprimento = 2, largura = 3, altura = 4
```

- Na chamada de uma função com dois ou mais argumentos padrão,
  - se o argumento omitido não estiver mais à direita, todos os argumentos a sua direita devem ser omitidos.
  - Ex. volume(2, ,4) **ERRO**

PASSAGEM DE PARÂMETROS PARA FUNÇÕES

- Passagem por valor, a função chamada recebe uma cópia do valor do argumento e trabalha exclusivamente com essa cópia.
  - Alterações na cópia da função não afetar o valor da variável original no chamador.

- • Com passagem por referência, a função chamada pode acessar o valor do argumento no chamador diretamente e modifique o valor se for mutável.

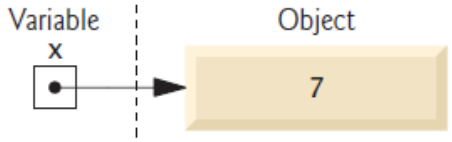
- Os argumentos Python são sempre passados por referência
  - Funções frequentemente manipulam objetos grandes - copiá-los frequentemente consumiria grandes quantidades de memória do computador e desempenho do programa significativamente lento.

PASSAGEM DE PARÂMETRO

- A atribuição

```
In [1]: x = 7
```

- Faz x referir-se a (ou "apontar para") o objeto inteiro contendo 7.



- A função id pode ser usada para obter um valor int único que identifica apenas aquele objeto enquanto ele permanece na memória

```
In [2]: id(x)
Out[2]: 4350477840
```

MOSTRANDO A REFERÊNCIA

```
In [18]: x = 7

In [19]: id(x)
Out[19]: 140728792949328

In [20]: def cube(number):
    ...:     print('id(number):', id(number))
    ...:     print('number is x:', number is x)
    ...:     return number ** 3
    ...:
   # argumento x e o parâmetro number referem-se ao mesmo objeto

In [21]: cube(x)
id(number): 140728792949328
number is x: True
Out[21]: 343

In [22]: x
Out[22]: 7
```

Operador **is** retorna True se seus dois operandos tiverem a mesma identidade

OBJETOS IMUTÁVEIS

- Quando uma função recebe como argumento uma referência a um objeto imutável mesmo que você tenha acesso direto a o objeto original no chamador, você não pode modificar o valor do objeto imutável original.

```
In [25]: def cube(number):
    ...:     print('id(number) before modifying number:', id(number))
    ...:     number **= 3 # cria novo objeto e atribui sua referencia a number
    ...:     print('id(number) after modifying number:', id(number))
    ...:     return number

In [26]: cube(7)
id(number) before modifying number: 140728792949328
id(number) after modifying number: 2116376523952
Out[26]: 343
```

OBJETOS MUTÁVEIS E IMUTÁVEIS

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

PASSAGEM DE PARÂMETROS POR VALOR E POR REFERÊNCIA

- *Em muitas linguagens, é possivel determinar como um parametro será passado a uma função.*
- *Passagem por valor – quando um argumento é passado por valor uma cópia deste é criada e passada a função*

- Passagem por referência – é dada à função a permissão de acessar o argumento diretamente, podendo modificá-lo
  - aumenta performance diminuindo o número de cópia de dados;
  - dimuni a segurança do dado

PASSAGEM DE PARÂMETROS POR VALOR E POR REFERÊNCIA

- Em Python, argumentos são sempre passados por referência ao objeto.
- Na prática, passagem por referência ao objeto pode ser visto como uma combinação da passagem por valor e por referência.
- Se uma função recebe uma referência a um objeto mutável, a função pode alterar o valor original deste objeto
  - Uma lista ou um dicinonário.
- Já se a função recebe uma referência a um objeto imutável, esta não pode alterar o seu valor original
  - Um número, uma string ou uma tupla

INTRODUÇÃO À CIÊNCIA DE DADOS: MEDIDAS DE DISPERSÃO

- Quando estamos falando sobre um grupo, o grupo inteiro é chamado de **população**. As vezes uma população é muito grande, como as pessoas que provavelmente votarão na próxima eleição presidencial dos EUA eleição, que é um número superior a 100 milhões de pessoas.

- Por razões práticas, as organizações de pesquisa tentando prever quem se tornará o próximo presidente seleciona pequenos subconjuntos da população conhecidos como **amostras**.

VARIÂNCIA

- Medidas de dispersão (também chamadas de medidas de variabilidade) ajudam a entender como os valores estão espalhados.

- Por exemplo, em uma classe de alunos, pode haver vários alunos cuja altura é próxima da média, com menor número de alunos que são consideravelmente mais baixo ou mais alto.

CALCULANDO A VARIÂNCIA

- Considere o cálculo manual da variância usando a seguinte população de 10 rodadas de dados de seis lados:

1, 3, 4, 2, 6, 5, 3, 4, 5, 2

VARIÂNCIA

- Para determinar a variância, começamos com a média desses valores - 3,5.
- Em seguida, subtraímos a média de cada valor de dado (isso produz alguns resultados negativos):

-2,5, -0,5, 0,5, -1,5, 2,5, 1,5, -0,5, 0,5, 1,5, -1,5

- Em seguida, elevamos ao quadrado cada um desses resultados (produzindo apenas positivos):

6,25, 0,25, 0,25, 2,25, 6,25, 2,25, 0,25, 0,25, 2,25, 2,25

- Finalmente, calculamos a média desses quadrados, que é 2,25 (22,5 / 10) - esta é a variância da população.

VARIÂNCIA

- O código a seguir usa o módulo ‘statistics’

- A função de variância para confirmar nosso resultado manual:

```
In [1]: import statistics
In [2]: statistics.pvariance([1, 3, 4, 2, 6, 5, 3, 4, 5, 2])
Out[2]: 2.25
```

DESVIO-PADRÃO

- O desvio padrão é a raiz quadrada da variância (neste caso, 1,5).
- Quanto menor for a variância e o desvio padrão, mais próximos os valores dos dados estão da média e menor é dispersão geral (ou seja, propagação).
- O código a seguir calcula o desvio-padrão populacional

```
In [3]: statistics.pstdev([1, 3, 4, 2, 6, 5, 3, 4, 5, 2])
Out[3]: 1.5
```

VANTAGEM DO DESVIO-PADRÃO

- Suponha que você tenha registrado as temperaturas de março em graus Celsius na sua área.
- Suponha que você tenha feito 31 medições, como 19, 32, 28 e 35.
  - As unidades desses números são graus.

- Quando você eleva ao quadrado suas temperaturas para calcular a variação da população, as unidades da população a variância torna-se "graus ao quadrado".

- Quando você tira a raiz quadrada da variância da população para calcular o desvio-padrão da população, as unidades tornam-se novamente graus, que são as mesmas unidades de suas temperaturas.

EXERCÍCIO

- “Adivinhe o número”. Escreva um programa que seleciona um número entre 1 e 1000 para ser adivinhado.
- O programa deve mostrar:

- Escolhi um número entre 1 e 1000. Você consegue adivinhar qual é ?

- O usuário deve entrar com os palpites até adivinhar o número.

- Coloque algumas frases para motivar o usuário.
  - **Parabéns, você acertou**
  - **Muito grande, tente novamente**
  - **Muito pequeno, tente novamente**
  - **Passou perto!**

LIERS' DICE

- No *Liers' dice*, cada jogador recebe 5 dados. Em cada rodada os jogadores lançam os seus dados, mantendo suas faces escondidas dos demais jogadores.
- Cada jogador então escolhe uma face (1 a 6) e da um palpite de quantas faces iguais a esta há na mesa.
- O próximo a jogar tem três alternativas
  - escolhe uma face e da um palpite, que deve ser necessariamente maior que o palpite anterior e passa para o próximo jogador.
  - chama o jogador anterior de mentiroso, neste caso abrem-se os dados e verifica-se o palpite corrente é verdadeiro. Se for o jogador atual perde um dado, senão o jogador anterior perde um dado.
  - concorda com o palpite do jogador anterior. Neste caso, abrem-se os dados e se o palpite estiver correto, o jogador anterior perde dois dados.
- Ganha o último jogador com dados na mesa.

EXERCÍCIO

- Implemente o Liers' dice.
  - O jogo deve ser para um usuário contra até 4 jogadores controlados pelo computador
  - Use funções;
  - Considere usar probabilidades para que um jogador do PC, de um palpite, chame de mentiroso ou concorde com o jogador anterior.
    - Use diferentes probabilidade para diferentes jogadores para simular diferentes comportamentos.
  - É possível utilizar-se de alguma heurística para melhorar os jogadores do PC?
    - Quem fica mais tempo na mesa, um jogador com heurística ou um jogador aleatório?