Atualizada em abr. 2021

Introdução a Python

Grupo de Estudo - PyLadies Brasil

pyladies Brasil

Encontros


Capítulo 1


Capítulo 2


Capítulo 3


Capítulo 4


Capítulo 5


Capítulo 6


Capítulo 7

Encontro 1

Mulher inspiradora

Nina trabalha pela inserção de mulheres na tecnologia e usa suas habilidades de hacker para lutar contra o racismo e o sexismo. Ativista da ciência da computação, descomplicando a tecnologia para facilitar a inclusão digital.

Além de apresentadora do canal Computação Sem Caô no YouTube, Nina também produz e apresenta o podcast Ogunhê, trazendo cientistas africanos. Além disso tudo, também está envolvida no Utopia, aplicativo de educação à distância.

Agenda

- Por que aprender Python?
- Variáveis, declaração e tipos
- Operadores aritméticos e relacionais
- Indentação
- Entrada e Saída
- Comentário
- Docstring

Por que aprender Python?

Python é uma linguagem com **sintaxe simples.**

Possui Código Aberto e a comunidade é extremamente engajada, como você pode perceber com as **PyLadies.**

É uma linguagem de **propósito geral**, ou seja, pode ser utilizada para motivações diversas: para desenvolvimento web ou aplicações científicas, por exemplo.

Muitas **oportunidades no mercado.**

Por que aprender Python?

Sendo Python uma **linguagem interpretada**, ela pode ser executada de duas formas: no modo interativo e não-interativo.

Com o **modo interativo**, é possível fazer algumas operações simples que serão interpretadas linha a linha diretamente do terminal.

O **modo não-interativo** é a forma clássica de se programar: basta criar o código e executar.

PEP - Python Enhancement Proposals

PEP, **Python Enhancement Proposals** [Propostas para Melhoramento no Python], como o nome diz, são propostas de aprimoramento ou de novas funcionalidades para a linguagem. Qualquer um pode escrever uma proposta. A comunidade Python testa, avalia e decide se deve ou não fazer parte da linguagem. Caso aprovado, o recurso é liberado para as próximas versões.

Aqui você pode encontrar todas as PEPs existentes.

Variáveis

De forma simplificada, variável é um nome que associamos a um valor ou expressão (informação).

- Sintaxe:

  ```
  variavel = conteudo
  ```

- Convenções para nomenclatura:
  - usar **algarismos, letras ou _**
  - **nunca começar com um algarismo**
  - **não usar palavras reservadas** do Python, como if, while, etc.

Declaração e atribuição

Python é uma linguagem **dinâmica** e não é necessária nenhuma declaração e especificação prévia para utilização de uma variável, ela será declarada durante sua **primeira atribuição**.

Python aceita atribuições encadeadas e mais de uma atribuição por vez usando vírgula.

Por ser uma linguagem dinâmica, as variáveis **não são associadas a tipos de dados**, seu tipo associado é diretamente o tipo de dado que armazena naquele momento, e pode variar no decorrer do código.

Declaração e atribuição

Exemplo de entrada	Exemplo de saída
`a = 10`	`>>> a` `10`
`a = 10` `a = "PyLadies"`	`>>> a` `'PyLadies'`

`inicio = fim = 0`	`>>> inicio`
	`0`
	`>>> fim`
	`0`

`capitulo, local = "PyLadies Floripa", "Florianópolis"`	`>>> capitulo`
	`'PyLadies Floripa'`
	`>>> local`
	`'Florianópolis'`

Exercício

Declare uma variável a, contendo o valor “PyLadies” e uma variável b contendo o valor 10.

Troque os conteúdos das variáveis, isto é, faça a receber o valor de b e b receber o valor de a.

Resolução

```
a = "PyLadies"
b = 10
a, b = b, a
```

Tipos Básicos de Dados em Python

Tipo de dado	Descrição
`bool`	Valores booleanos: `True` ou `False`
`float`	Valores com ponto flutuante: `1.0`, `-2.3`, `3.5` ...
`int`	Valores inteiros: ..., `-2`, `-1`, `0`, `1`, `2`, `3`, ...
`complex`	Números complexos: `8+1j`, `2-3j`, `1000+1j` ...
`str`	Textos: python considera texto tudo que estiver entre aspas (simples ou duplas): `"PyLadies Brasil"`, `"Python"`, `"(00)00000-0000"`, ...

Tipos de dados - Type

Como Python é uma linguagem dinâmica e tem seu tipo de dado determinado pelo seu conteúdo corrente. A função type exibe o tipo.

Exemplo de entrada	Exemplo de saída
`type(True)`	`<class 'bool'>`
`type(3.14)`	`<class 'float'>`
`type(10)`	`<class 'int'>`
`type(5+2j)`	`<class 'complex'>`
`type("PyLadies")`	`<class 'str'>`

Operadores aritméticos

Operador	Nome	Exemplo	Saída
+	Adição	`39 + 13`	`52`
-	Subtração	`39 - 13`	`26`
*	Multiplicação	`39 * 13`	`507`
/	Divisão	`39 / 13`	`3.0`
//	Divisão Inteira	`39 // 13`	`3`
%	Resto da Divisão	`39 % 13`	`0`
**	Potenciação	`39 ** 13`	`482880748567480579719`

Operadores relacionais

Operador	Nome	Exemplo	Saída
>	Maior	`"Python" > "Outras Linguagens"`	`True`
<	Menor	`50 < 5`	`False`
>=	Maior ou igual	`10 >= 5 + 5`	`True`
<=	Menor ou igual	`10 + 1 <= 5 + 5`	`False`
==	Igual	`1 == 1 + 0`	`True`
!=	Diferente	`"Python" != "Outras Linguagens"`	`True`

Precedência e associatividade

Precedência indica qual o operador calculado primeiro, e associatividade, a ordem dos cálculos de operadores de mesma precedência. Tabela listada por prioridade:

Operador	Nome	Associatividade
`()`	Parênteses	Da esquerda para a direita
`**`	Potenciação	Da direita para a esquerda
`+, -`	Positivo e negativo unário	Da direita para a esquerda
`*, /, //, %`	Multiplicação, divisão, divisão inteira e resto	Da esquerda para a direita
`+, -`	Adição e subtração	Da esquerda para a direita

Incremento, decremento e demais operações

Operações que alteram o valor de uma variável e salva nela mesma.

Incremento

```
x += 1 #mesma coisa que: x = x+1
```

Decremento

```
x -= 3 #mesma coisa que: x = x-3
```

Obs: funciona para qualquer operação (+, -, *, %, **, //):

Indentação

A indentação em Python é **obrigatória, pois ela delimita os blocos de código**. Foi elaborada com o objetivo de deixar os programas mais legíveis.

Ela é representada por 4 espaços vazios ou um tab.

Exemplo de utilização:

```
if 10 > 5:
    print(f'O número 10 é maior do que 5.')
else:
    print(f'O número 10 não é maior do que 5.')
```

Exercício

No interpretador interativo, IDLE (Windows) ou Terminal (Linux):

- Crie uma variável intitulada “trecho” e atribua um dado do tipo str;
- Armazene na variável “qnt_repeticao” o valor 5;
- Na variável “texto_completo”, armazene a multiplicação da primeira variável com a “qnt_repeticao”;
- Na variável “comparacao”, armazene o resultado da comparação de igualdade entre as variáveis “texto_completo” e a multiplicação de “trecho” e “qtd_repeticao”.

Resolução

```
trecho = "PyLadies são demais! "
qtd_repeticao = 5
texto_completo = trecho * qtd_repeticao
comparacao = texto_completo == trecho * qtd_repeticao
```

Entrada e saída

Entrada:

```
input("Texto opcional que será exibido ao usuário")
```

Saída:

```
print(f"Textos a serem exibidos e {variaveis}")
```

OBS: Note que as **variáveis são impressas entre chaves** e que há um "**f**" no começo do texto, que é denominada **f-string**.

Entrada

Python considera **todas as entradas como texto, cada linha digitada como um único conteúdo contínuo**.

Suponha que você digite numa mesma linha seu nome, sua idade e grupo PyLadies de origem, Python reconhecerá tudo como uma mesma entrada de texto, sem divisões.

Caso você deseje um tipo diferente, será necessário converter explicitamente. Exemplo de utilização:

```
variavel_convertida = int (variavel)
```

Conversão

Tipo	Comando de conversão
Booleano (**bool**)	`bool (variavel)`
Inteiro (**int**)	`int (variavel)`
Real (**float**)	`float (variavel)`
Texto (**str**)	`str (variavel)`
Números complexos (**complex**)	`complex (variavel)`

Entrada

As operações de leitura e conversão de dados podem ser **compostas**, como por exemplo:

```
comunidade = int(input('Digite o número de PyLadies: '))
```

Mas há uma restrição, a conversão **só pode ser aplicada a um valor por vez**! E em Python, **todos os valores digitados em uma mesma linha serão considerados o mesmo texto**, portanto por hora use um valor por linha, ensinaremos como contornar esse problema em breve!

Exercício

Usando apenas um print(), imprima as variáveis abaixo:

- trecho
- qnt_repeticao
- texto_completo
- comparacao

Resolução de Exercício

```
print(f'O trecho "{trecho}" repetido {qnt_repeticao} fica assim: \n "{texto_completo}". \n Os textos são iguais? \n {comparacao}')
```

Exercício

Faça um programa que receba o número de camisetas que devemos mandar fazer para o PyLadies, e imprima quanto de entrada teremos que pagar.

Considere que cada camiseta custa R$25,00, e a entrada para o fabricante é de 50% do valor total.

Resolução

```
valor = 25
quantidade = int(input("Digite o número de camisetas que serão confeccionadas: "))
entrada = (valor * quantidade)/2
print(f"O valor da entrada a ser paga para o fabricante é de R$ {entrada:.2f}")
```

Resolução

```
valor = 25
quantidade = int(input("Digite o número de camisetas que serão confeccionadas: "))
entrada = (valor * quantidade)/2
print(f"O valor da entrada a ser paga para o fabricante é de R$ {entrada:.2f}")
```

Máscara de formatação números decimais

Comentário

Em Python, um comentário deve explicar o que aquela linha ou trecho de código está fazendo, para que ajude a outras pessoas entenderem o código.

Sintaxe:

```
# comentário de 1 linha


'''
<comentário com várias linhas>
'''
```

Docstring

Usada para documentar o código, deve vir logo após a definição de uma classe ou função.

Sintaxe:

```
'''
<comentário com várias linhas>
'''
```

Podemos acessar a docstring através de __doc__

Para você praticar em casa :)

O PyLadies ministrará um minicurso aberto à comunidade e graças ao grupo de Relações Externas, conseguimos um patrocínio para o coffee break.

Leia o número de participantes do minicurso e imprima: quantos litros de refrigerante teremos que comprar, quantas coxinhas e quanto iremos gastar.

Considere que cada participante toma 400ml de refrigerante, come 10 coxinhas. Cada litro de refrigerante custa R$4,00 e cada coxinha custa R$0,50.

Continue estudando...

Lista de exercícios no HackerRank

Série no Youtube “Introdução a Python: para mulheres que sabem programar em outra linguagem”

e-Book “Introdução a Python: para mulheres que sabem programar em outra linguagem”

Encontro 2

Mulher inspiradora

Katie Bouman e o time que ela liderava conseguiram pela primeira vez reconstituir uma foto de um buraco negro em 2019 a partir de algoritmos utilizando Python!

Bibliotecas utilizadas no projeto: Numpy, Scipy, Pandas, Jupyter, Matplotlib, Astropy

Vídeo: How to take a picture of a black hole | Katie Bouman [TED] [legendado em pt-br]

Artigo: First M87 Event Horizon Telescope Results. III. Data Processing and Calibration [em inglês]

Repositório Github: ehtim (eht-imaging) [em inglês]

Agenda

- Resolução de exercício
- Strings
- Operadores Lógicos
- Estruturas Lógicas e Condicionais

Resolução

```
participantes = input('Digite a quantidade de participantes: ')
participantes = int(participantes)
refrigerante_litros = 0.4 * participantes
refrigerante_preco_total = refrigerante_litros *4
coxinha_quantidade = 10 * participantes
coxinha_preco_total = 0.5 * coxinha_quantidade
custo_total = coxinha_preco_total + refrigerante_preco_total

print(f'Serão necessários {refrigerante_litros} litros de
refrigerante. \nSerão necessárias {coxinha_quantidade}
cozinhas.\nIremos gastar R$ {custo_total:.2f}.')
```

Strings

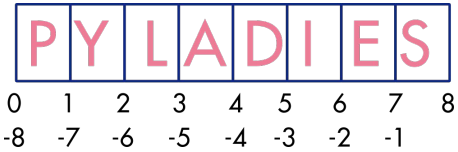
Python é uma linguagem bastante reconhecida pelos recursos que oferece para processamento de textos. Strings são conjuntos de caracteres definidos **entre aspas simples ou duplas**. Para declarar strings com mais uma linha, usa-se três aspas em sequência.

```
"String de uma única linha"
'''
String com mais de
uma linha
'''
```

Strings

Operador	Explicação	Exemplo	Saída
Concatenação (**+**)	**Cria uma nova string com a concatenação das duas strings passadas**	`"Amo" + "Python"`	`"AmoPython"`
Repetição (*****)	**Cria uma nova string com *n* repetições da string original**	`"PyLadies <3 "*3`	`"PyLadies <3 PyLadies <3 PyLadies <3 "`
Indexação (**[]**)	**Seleciona um caracter de uma string** (inicia em 0)	`"PyLadies"[1]`	`"y"`

Strings



Operador	Explicação	Exemplo	Saída

Fatiamento (`[::]`)	**Seleciona substring iniciando na posição de início, até o fim (não incluso), com passo n**.
	Se não for definido início, iniciará no primeiro caracter.
	Se não for definido fim, irá até o último caracter.
	Se não for definido um passo, será 1.
		`"PyLadies Brasil"[2:8:1]`
		`"Sem início"[:3]`
		`"Sem final"[4:]`
		`"Sem fim e com passo 2"[10::2]`
		`"Com passo negativo"[8:3:-1]`
			`"Ladies"`
			`"Sem"`
			`"final"`
			`"cmpso2"`
			`"ossap"`

Strings

Operador	Explicação	Exemplo	Saída
Verificação (`in`)	Verifica se **há determinada substring** na string original	`"Ladies" in "PyLadies Brasil"`	`True`
Verificação negativa (`not in`)	Verifica se **não há uma substring** na string original	`"Py" not in "Python"`	`False`
Tamanho (`len`)	Retorna o **tamanho** de uma string	`len("PyLadies Brasil")`	`15`

Strings

Função	Explicação	Exemplo	Saída
`"string".startswith("substring")`	Retorna verdadeiro se a string **inicia com a substring passada,** senão falso	`"PyLadies Macaé".startswith("Py")`	`True`
`"string".endswith( "substring")`	Retorna verdadeiro se a string **termina com a substring passada,** senão falso	`"PyLadies Natal".endswith("Recife")`	`False`
`"string".replace("antigo", "novo", limite_opcional)`	**Substitui a string original pela nova string**, com limite de substituições (se não houver, substitui todas as ocorrências)	`"PyLadies Natal e Natal".replace("Natal", "Recife", 1)`	`"PyLadies Recife e Natal"`

Strings

Função	Explicação	Exemplo	Saída
`"string".count("substring")`	Retorna **número de ocorrências** da substring na string original	`"PyLadies amam Python".count("Py")`	`2`
`"string".index("substring")`	Retorna a **posição** onde a substring inicia na string original (buscando da esquerda para a direita) ou **ValueError** se não for encontrada.	`"PyLadies São Carlos".index("São")`	`9`
`"string".rindex("substring")`	Retorna a **posição** onde a substring inicia na string original (buscando da direita para a esquerda) ou **ValueError** se não for encontrada.	`"PyLadies amam Python".rindex("Py")`	`14`

Strings

Função	Explicação	Exemplo	Saída
`"string".find("substring")`	Retorna a **posição** onde a substring inicia na string original (buscando da esquerda para a direita) ou **-1** se não for encontrada.	`"PyLadies São Carlos".find("Brasil")`	`-1`
`"string".rfind("substring")`	Retorna a **posição** onde a substring inicia na string original (buscando da direita para a esquerda) ou **-1** se não for encontrada.	`"PyLadies Paraíba dominando o mundo".rfind("ndo")`	`31`

Strings

Função	Explicação	Exemplo	Saída
`"string".isalnum()`	Retorna **verdadeiro se a string é composta exclusivamente por letras e dígitos**	`"Mudar123".isalnum()`	`True`
`"string".isalpha()`	Retorna **verdadeiro se a string é composta exclusivamente por letras**	`"PyLadies Manaus".isalpha()`	`False`
`"string".isdigit()`	Retorna **verdadeiro se a string é composta exclusivamente por digitos**	`"10".isdigit()`	`True`
`"string".isspace()`	Retorna **verdadeiro se a string é composta exclusivamente por caracteres espaço**	`"\n\t ".isspace()`	`True`

Strings

Função	Explicação	Exemplo	Saída
`"string".capitalize()`	Seta a **primeira letra como maiúscula** e as demais minúsculas	`"pYtHon é a melhor LINGUAGEM <3".capitalize()`	`"Python é a melhor linguagem <3"`
`"string".title()`	Seta **toda primeira letra de cada palavra maiúscula** e as demais minúsculas	`"pyladies brasil".title()`	`"Pyladies Brasil"`
`"string".swapcase()`	**Inverte os cases de todas as letras**	`"pYlADIES bRASIL".swapcase()`	`"PyLadies Brasil"`

Strings

Função	Explicação	Exemplo	Saída
`"string".lower()`	Seta todas as **letras minúsculas**	`"Python é LINDOO!".lower()`	`"python é lindoo!"`
`"string".upper()`	Seta todas as **letras maiúsculas**	`"amo python!".upper()`	`"AMO PYTHON!"`
`"string".center(numero, "caracter opcional")`	**Centraliza uma string entre N caracteres, preenchendo os demais com o caracter passado** (se não for passado, será espaço)	`"Py <3".center(9, "*")` `":D".center(6)`	`"**Py <3**"` `"  :D  "`

Strings

Função	Explicação	Exemplo	Saída
`"string".zfill()`	**Preenche com zeros à esquerda** até completar n caracteres	`"Py".zfill(5)`	`"000Py"`
`"string".rjust(numero, "caracter opcional")`	**Alinha n caracteres à direita** preenchendo a string com o caracter passado (ou espaços, na ausência deste)	`"Py".rjust(5, "-")`	`"Py---"`
`"string".ljust(numero, "caracter opcional")`	**Alinha n caracteres à esquerda** preenchendo a string com o caracter passado (ou espaços, na ausência deste)	`"Py".rjust(5, "+")`	`"+++Py"`

Strings

Função	Explicação	Exemplo	Saída
`"string".split("string de separação opcional")`	**Divide a string em várias substrings a partir de uma string de separação.** Se não for passada string de separação, usará espaços.	`"PyLadiesPyBar".split("Py")` `"PyLadies Brasil".split()`	`['', 'Ladies', 'Bar']` `['PyLadies', 'Brasil']`
`"string".strip("string opcional")`	**Remove substrings do início e final da string.** Se não for passada string, removerá **espaços em branco**.	`"PyPyLadies PyLadiesPy".strip("Py")` `"  Python".strip()`	`"Ladies PyLadies"` `"Python"`

Exercício

Toda vez que o Grupo de Trabalho de Documentação tenta fazer um levantamento de capítulos do Brasil temos um mesmo problema: alguns capítulos colocam apenas o local e outros colocam “PyLadies” também no nome do capítulo, em letra maiúscula ou minúscula.

Como uma pessoa muito organizada e que sabe Python você resolveu dar um basta nessa bagunça: vai **garantir que todos os capítulos tenham “PyLadies” no início nome**, obviamente, tomando o cuidado de não colocá-lo duas vezes.

Use as funções de string para, dado um nome, retorná-lo no formato correto!

Resolução

```
nome = input("Digite o nome do capítulo: ")
# a função replace só substitui valores existentes
# se o nome não tiver "PyLadies " no nome, nada ocorrerá
nome = nome.lower().replace("pyladies", "")
# como removemos "PyLadies", podem ter sobrado espaços
# então vamos removê-los com a função strip
nome = nome.strip()
# setando cada nome com letra inicial maiúscula
nome = nome.title()
print(f"PyLadies {nome}")
```

Split - Entrada de dados

Ainda se lembra que Python lê **dados de entrada como uma única linha de texto**? Por exemplo: como você extrairia um nome e uma idade digitados numa mesma linha?

Como todas as entradas do Python são consideradas textos, portanto strings, **você pode utilizar todas as funções de string sobre elas**, inclusive a função split()!

```
nome, idade = "Ana 27".split()
```

Você pode criar quantas variáveis sejam necessárias para armazenar seus dados. Lembre-se de **converter** o que for necessário depois!

Exercício

As PyLadies do Grupo de Trabalho de comunicação estão enviando e-mails para cada capítulo e queriam transformar uma informação, para isso pediram a sua ajuda!

Elas tem as informações **e-mail e ano de criação do capítulo, e gostariam de ter e-mail e idade do capítulo**. Considere o ano como 2021 e retorne a informação atualizada.

Resolução

```
email, ano_de_criacao = input("Digite e-mail e ano de
criação do capítulo: ").split()
ano_de_criacao = int(ano_de_criacao)
print(f"{email} {2021 - ano_de_criacao}")
```

Operadores Lógicos

Os operadores lógicos (também conhecidos como operadores booleanos) são utilizados para auxiliar a criar estruturas de decisão.

Em Python eles são escritos por extenso (em inglês):

Operador booleano	Python
e	**and**
ou	**or**
negação	**not**

Precedência

Precedência indica qual o operador calculado primeiro, listadas por ordem de prioridade.

Todas as expressões serão avaliadas antes dos operadores lógicos.

Operador	Nome
`()`	Parênteses
`not`	Negação
`and`	E
`or`	Ou

Operadores Lógicos

Exemplo de entrada	Exemplo de saída
`True and False`	`False`
`num = 100` `nume % 2 == 0 and num % 5 == 0 and 0 <= num <= 100`	`True`
`1+2+3+4+5+6 == 7*(7-1)//2 and 7*(7-1)//2 % 3 == 0`	`True`
`num = 25` `num % 5 == 0 or num % 2 == 0 and 10 <= num <= 20`	`True`
`num = 36` `num % 4 == 0 and num % 6 == 0 and not num % 8 == 0`	`True`

Operadores Lógicos

Os valores considerados falsos em Python são:

Constantes	Números zero	Sequências vazias
`False`	`int (0)`	`list ( [] )`
`None`	`float (0.0)`	`tuple ( () )`
	`complex (0j)`	`dict ( {} )`
		`set ( )`
		`str ( "" )`
		`range ( 0 )`

Exercício

Juliana estava pensando se deveria entrar para as PyLadies e resolveu consultar sua amiga Amanda, sobre a possibilidade de ingressar. Amanda explicou para Juliana que para ser membra das PyLadies era necessário se identificar como **mulher** e **ter vontade de aprender**.

Juliana ficou animada com a resposta e resolveu criar seu primeiro programa em Python para analisar quais pessoas podem ingressar no PyLadies. Agora que já conhecemos os operadores lógicos, que tal você criar um programa sobre essa possibilidade também?

Resolução

```
eh_mulher = True
tem_vontade = True
pode_participar = eh_mulher and tem_vontade
print(pode_participar)
```

Estruturas Lógicas e Condicionais

```
if condicao:
    codigo
```

```
if condicao:
    codigo
else:
    codigo
```

```
if condicao:
    codigo
elif condicao:
    codigo
else:
    codigo
```

A indentação é muito importante, pois é o que define blocos de código em Python!

Estruturas Lógicas e Condicionais

Python preza pela legibilidade, por isso existe a cláusula **elif**. Ela substitui um else com um if encadeado, os quais gerariam **aninhamento**:

```
if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
    if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
        $user = read_user($_POST['user_name']);
        if (!isset($user['user_name'])) {
            if ($_POST['user_email']) {
                if (strlen($_POST['user_email']) < 65) {
                    if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                        create_user();
                        $_SESSION['msg'] = 'You are now registered so please login';
                        header('Location: ' . $_SERVER['PHP_SELF']);
                        exit();
                    } else $msg = 'You must provide a valid email address';
                } else $msg = 'Email must be less than 64 characters';
            } else $msg = 'Email cannot be empty';
        } else $msg = 'Username already exists';
    } else $msg = 'Username must be only a-z, A-Z, 0-9';
} else $msg = 'Username must be between 2 and 64 characters';
```

Queremos evitar isso!

Exercício

Para ministrar as aulas de Introdução a Python das PyLadies Brasil serão necessárias ao menos **quatorze** PyLadies muito proativas, que implementarão a primeira fase do nosso grupo de estudos!

Você ficou encarregada de escrever um programa para **checar se já temos voluntárias suficientes**, pois não quer correr o risco de esquecer o número de pessoas necessárias, e ter que fazer a distribuição toda novamente.

Se o número for suficiente, imprima "Sucesso", caso contrário, imprima "Precisamos de mais n voluntárias!", onde n é o número mínimo de voluntárias que ainda precisamos encontrar.

Resolução

```
pyladies = int(input("Digite o número de voluntárias: "))
if pyladies >= 14:
    print("Sucesso!")
else:
    print(f"Precisamos de mais {14-pyladies} voluntárias!")
```

Para você praticar em casa :)

A comunidade PyLadies Brasil no Slack está crescendo a cada dia! Para recepcionar as novas membras, você foi chamada para fazer um programa para enviar “boas-vindas” para todas as Ladies no Slack.

Pergunte o nome e se já é membra de um capítulo do PyLadies e **receba** o nome e a resposta da membra.

Se a resposta da membra for ‘sim’, **imprima**: “Olá [nome da Lady], bem-vinda ao PyLadies Brasil!”

Se a resposta for ‘nao’, **imprima**: “Olá [nome da Lady], vamos procurar seu capítulo juntas ;D”

Se for qualquer outra resposta, **imprima**: “Olá [nome da Lady], já conhece o PyLadies? :D”

Continue estudando...

Lista de exercícios no HackerRank

Série no Youtube “Introdução a Python: para mulheres que sabem programar em outra linguagem”

e-Book “Introdução a Python: para mulheres que sabem programar em outra linguagem”

Encontro 3

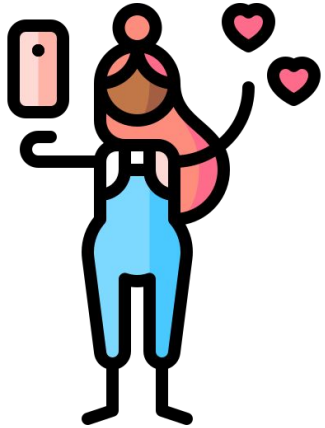
Mulher inspiradora

Fernanda Wanderley Doutora em Inteligência Computacional, interessada em aplicações de aprendizado de máquina em problemas biomédicos. Trabalha com detecção de patologias em imagens de raio-x e sangramentos em tomografia de tórax, e também já pesquisou sobre previsão de eficácia de quimioterapia em câncer de mama e busca de variantes genéticas em DNA.

Sua maior satisfação é impactar positivamente a vida das pessoas.

Agenda

- Resolução de exercício
- Range
- Estruturas de Repetição



Resolução

```
nome = input("Olá Lady, qual seu nome? ")
capitulo = input("Já faz parte de algum capítulo? ").lower()

if capitulo == 'sim':
  print(f"Olá, {nome}! Bem-vinda ao PyLadies Brasil!")
elif capitulo == 'nao':
  print(f"Olá, {nome}! Vamos procurar seu capítulo juntas ;D")
else:
  print(f"Olá, {nome}! Já conhece o PyLadies? :D")
```

Range

A função **range() gera um iterável com N elementos, a partir de parâmetros de início, final e passo fornecidos**. Seu formato geral é:

```
range(inicio,fim,passo)
```

Range gera um conjunto de dados começando em início, terminando em fim, e iterando de passo em passo.

Mas também pode ser escrita como:

```
range(inicio,fim)
range(fim)
```

Range

Na função **range**, se for passado apenas **um parâmetro**, a função considerará como o parâmetro **fim**, se forem passados **dois parâmetros**, como **início e fim**, por padrão.

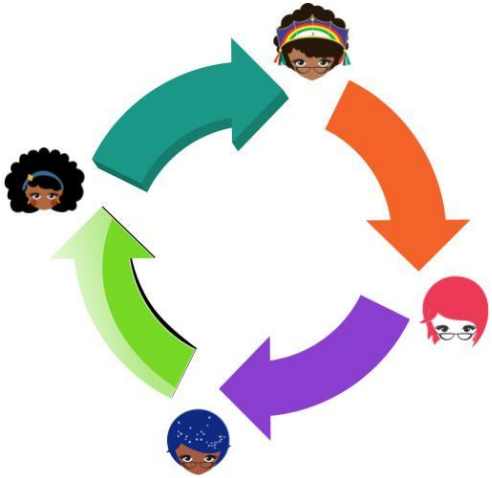
Início e **passo** são parâmetros não-obrigatórios, e recebem como valores padrão **0** e **1**, respectivamente, como padrão caso não sejam passados.

Fim é um limitante não-inclusivo, isto é, o último elemento será **estritamente menor que fim**.

Estruturas de Repetição (laços ou loops)

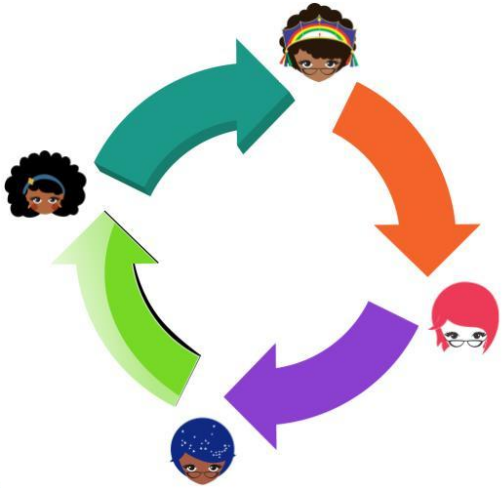
Executam um conjunto de comandos repetidamente satisfazendo a determinada condição.

As duas estruturas de repetição mais comuns são **for** e **while.**



While

```
while condição:
    comandos
```



```
i = 1
while i <= 10:
    print(f'Oii, Lady {i}!')
    i += 1
```

Exercício

Dandara, uma das nossas PyLadies mais animadas, tem uma meta bastante audaciosa: ter 1000 integrantes no grupo do Telegram das PyLadies Brasil!

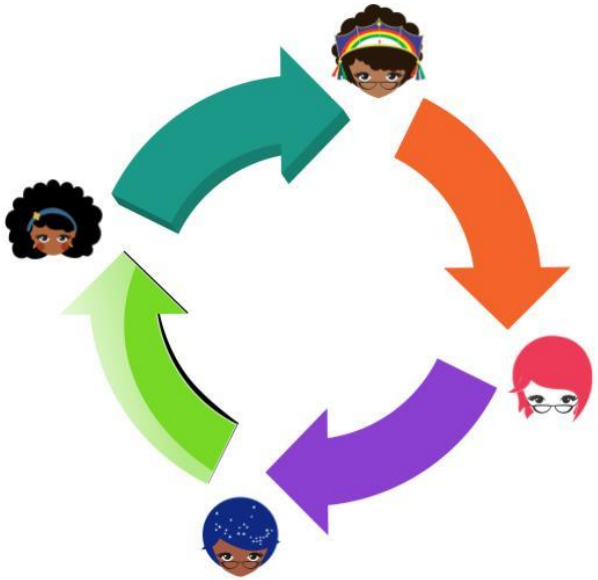
Você está bastante ansiosa para saber se já atingimos a meta, por isso resolveu fazer um programa em Python que recebe o número de integrantes atual continuamente, e, **enquanto não houver 1000 mulheres ou mais**, retorna a mensagem “Vamos levar a Pylavra do Python por aí!”, e **quando o número lido for igual ou superior a 1000**, você encerra o programa imprimindo a mensagem “Sucesso! Hora de dobrar a meta!”. Contamos com você!

Resolução

```
mulheres_no_telegram = int(input())
while mulheres_no_telegram < 1000:
    print("Vamos levar a Pylavra do Python por aí!")
    mulheres_no_telegram = int(input())
print("Sucesso! Hora de dobrar a meta!")
```

For

```
for valor in sequência:
    código
```



```
for i in range(1, 11):
    print(f'Oii, Lady {i}!')
```

Exercício

O hobby das PyLadies é irem juntas para o bar, seja para tomar cerveja, suco, vinho, refrigerante ou suas cachaças preferidas.

Como o hobby é recorrente, resolveram marcar os bares **todas as sextas-feiras**, para deixarem essa data sempre reservada!

Você, que é uma PyLady muito entusiasmada com os rolês, resolveu fazer um programa que imprima todos os dias do mês de dezembro em que teremos PyLadies Bar!

O ano é 2021 e o mês de dezembro começa numa quarta-feira!

Resolução

```
for i in range(3,32,7):
    print(f"{i} de dezembro!")
```

Ou:

```
for i in range(0,32):
    if i % 7 == 3:
        print(f"{i} de dezembro!")
```

Exercício

Escreva um programa que imprima os números de 1 a 30, mas para múltiplos de 3 imprima “Py” ao invés do número, e para os múltiplos de 5 imprima “Ladies”. Para números que são múltiplos de 3 e 5, imprima “PyLadies”.

Escreva duas soluções, uma usando **while** e outra usando **for**.

Resolução utilizando for

```
for i in range(1, 31):
    if i % 3 == 0 and i % 5 == 0:
        print ('PyLadies')
    elif i % 3 == 0:
        print ('Py')
    elif i % 5 == 0:
        print ('Ladies')
    else:
        print (str(i))
```

Resolução utilizando while

```
i = 1
while i < 31:
    if i % 5 == 0 and i % 3 == 0:
         print('PyLadies')
    elif i % 3 == 0:
        print ('Py')
    elif i % 5 == 0:
        print('Ladies')
    else:
        print(i)
    i += 1
```

Exercício

O Grupo de Trabalho de Criação de Materiais está tentando escrever um e-mail fofinho para dar as boas-vindas ao nosso curso de introdução à Python, mas só temos o nome completo das participantes, e isso seria formal demais e nada fofinho!

Você ficou encarregada de criar um programa em Python que **selecione apenas o primeiro nome** de uma lista de nomes completos para que as outras integrantes possam enviar o e-mail fofinho :)

Dado um número **N de participantes, seguido por N linhas cada uma com um nome, retorne no formato especificado.** Dica: trate um nome por cada vez e já imprima o retorno.

Resolução

```
num_participantes = int(input())
for i in range(num_participantes):
    nome = input()
    primeiro_nome = nome.split()[0]
    print(primeiro_nome)
```

Exercício

As PyLadies Brasil estão criando stickers com *#hashtags* motivacionais para nossa terceira edição do PyLadies Conf Brasil, e a gráfica solicitou que enviássemos um arquivo com o **número de repetições exatas** para facilitar o seu trabalho, já que o estagiário de lá só sabe copiar e colar, e nós sabemos programar!

Você ficou responsável por gerar o arquivo final, tendo como base a ***#hashtag* de entrada e o número de PyLadies** que se inscreveram para o evento. PS: sempre faça o **dobro de stickers** porque todas as PyLadies amam stickers!

Resolução utilizando o comando for

```
# resolução com for
hashtag = input("Digite a hashtag: ")
pyladies = int(input("Digite o numero de PyLadies: ")) * 2
# vamos fazer o dobro de stickers necessários!
for sticker in range(pyladies):
    print(hashtag)
```

Resolução utilizando o comando while

```
# resolução com while
hashtag = input("Digite a hashtag: ")
pyladies = int(input("Digige o numero de PyLadies: ")) * 2
# vamos fazer o dobro de stickers necessários!
stickers = 0
while stickers < pyladies:
    print(hashtag)
    stickers += 1
```

Para você praticar em casa :)

Várias empresas procuram as PyLadies Brasil para propor parcerias, cursos, eventos, divulgação de vagas e tantas outras atividades.

Na última semana, uma empresa nos procurou para oferecer cursos de programação à **meninas de quatorze a dezesseis anos**, e você ficou responsável por determinar quantas garotas atendem a este requisito.

Dado **um número N de PyLadies e, em seguida, N linhas contendo idades de PyLadies, conte quantas satisfazem os requisitos** da parceria.

Continue estudando...

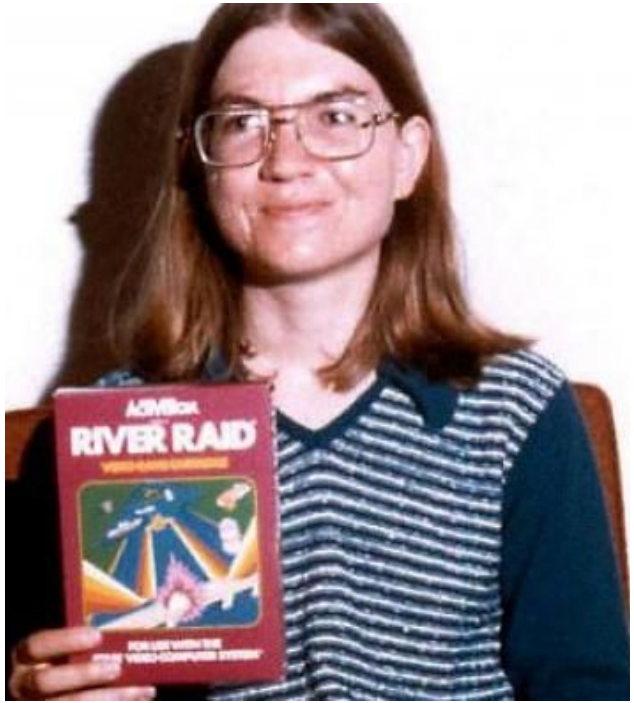
Lista de exercícios no HackerRank

Série no Youtube “Introdução a Python: para mulheres que sabem programar em outra linguagem”

e-Book “Introdução a Python: para mulheres que sabem programar em outra linguagem”

Encontro 4

Mulher inspiradora



Carol Shaw (1955 - *) é a primeira mulher a ser tornar funcionária de uma empresa de *games*, como a Activision e Atari. E entrou para história como pioneira pela luta da igualdade de gêneros no ramo tecnológico.

Agenda

- Resolução de exercício
- Break
- Continue
- While ... Else
- For ... Else

Resolução utilizando o comando for

```
# resolução com for
pyladies = int(input("Digite o número de PyLadies: "))
publico_alvo = 0
for pylady in range(pyladies):
    idade = int(input("Digite a idade desta PyLady: "))
    if 14 <= idade <= 16:
        publico_alvo += 1
print(f"{publico_alvo} PyLadies podem participar!")
```

Resolução utilizando o comando while

```
pyladies = int(input("Digite o número de PyLadies: "))
publico_alvo = 0
pylady = 0
while pylady < pyladies:
    idade = int(input("Digite a idade desta PyLady: "))
    if 14 <= idade <= 16:
        publico_alvo += 1
    pylady += 1
print(f"{publico_alvo} PyLadies podem participar!")
```

Break

E se você quiser sair do laço antes da condição de parada?

Tem solução para isso!!! Você pode usar break ;)

```
i = 1
while (i <= 10):
    if i > 2:
        break
    print(f'Oii, Lady {i}!')
    i += 1
```

```
for i in range(10):
    if i > 4:
        break
    print(f'Oii, Lady {i}!')
```

Exercício

Para movimentar os pybares online, as PyLadies resolveram que usariam jogos para interação das participantes, e aquela que **completasse 100 pontos ou mais pontos primeiro** ganharia uma camiseta do 1 ° PyLadies Conf que a Débora Azevedo tem guardada!

Você recebeu uma lista completa com todas as atualizações de pontuações da noite: cada uma das N linhas tem o user de uma PyLady e uma pontuação. Quando uma PyLady fazia outro ponto, seu user era inserido novamente na lista com a sua nova pontuação, portanto não é sua função fazer o somatório (isso já foi feito por outra pylady), você apenas precisa encontrar a primeira a fazer 100 pontos.

Exercício

Entrada:

```
8
anapaula 20
ceci 40
anadulce 60
anapaula 50
ceci 99
anadulce 101
anapaula 100
ceci 200
```

Saida:

```
anadulce
```

OBS: considere que os users serão escritos todos em letras minúsculas, sem acentuação e sem espaços.

Resolução com for

```
num_lista = int (input())
for i in range(num_lista):
  lady, pontos = input().split()
  pontos = int (pontos)
  if pontos >= 100:
      print(lady)
      break
```

Resolução com while

```
num_lista = int (input())
i = 0
while i <= num_lista:
    lady, pontos = input().split()
    pontos = int (pontos)
    if pontos >= 100:
        print(lady)
        break
    i += 1
```

Continue

E se você não quiser sair do loop, mas apenas “pular” alguns comandos? Use a cláusula **continue**! Ela permite que, ainda no meio do laço, você passe para a próxima iteração.

```
i = 1
while (i <= 10):
    i += 1
    if i == 2:
        continue
    print(f'Oii, Lady {i}!')
```

```
for i in range(1, 11):
    if i == 2:
        continue
    print(f'Oii, Lady {i}!')
```

Exercício

Percebemos recentemente que existem Anas demais nas reuniões do PyLadies Brasil, o que aumenta consideravelmente o número de equívocos e confusões.

Você foi incumbida de **redigir uma lista com o primeiro nome das participantes da reunião, mas, se o primeiro nome for Ana, você deve imprimir o segundo nome também**, para auxiliar na identificação.

Dado um número N de integrantes, as N seguintes linhas serão de nomes de PyLadies que você deverá redigir na sua lista de saída!

Resolução

```
num_participantes = int(input())
for i in range(num_participantes):
    nome = input()
    primeiro_nome = nome.split()[0]
    if primeiro_nome != "Ana":
        print(primeiro_nome)
        continue
    segundo_nome = nome.split()[1]
    print(f"{primeiro_nome} {segundo_nome}")
```

While... Else

```
while (condição):
    comandos
else:
    comandos
```

```
i = 0
while (i <= 10):
    i += 1
    if i == 5:
        break
    print(f'Oii, Lady {i}!')
else:
    print('Bem vindas!')
```

Exercícios

Durante a pandemia de covid-19, as PyLadies Brasil começaram a utilizar o Discord para suas reuniões, o único ponto é que o Discord **permite apenas 25 users** com câmera em uma sala, e às vezes superamos esse número.

Você fez um bot que monitora o canal de voz no Discord, e vai **avisar se alguma hora chegarmos a 24 pessoas** e printar a mensagem "Abandonando o Discord e partindo para o Meet!". **Se chegarmos à zero pessoas sem estourar esse limite (isto é, se a reunião terminar sem exceder este número)**, ele deve imprimir a mensagem "Reunião realizada com sucesso!" PS: a explicação continua na próxima página :)

Exercícios

Seu bot recebe mensagens pela API do Discord a **cada vez que um usuário entra ou sai do canal de voz**. Cada mensagem consiste em um **user** (uma cadeia de caracteres composta de letras maiúsculas e minúsculas, números, underscore e hífens sem espaços) e um **status**, que pode ser "**entrou**" ou "**saiu**" separados por um espaço.

Você deve ler as linhas de entrada **até que tenham 24 pessoas** simultâneas no canal, ou **até que tenham 0 pessoas no canal**, e tenha acabado a reunião. Considere que as entradas são consistentes, que só pode sair uma pessoa que entrou, e que os canais são fechados quando chega a 0 participantes. Dica: não se importe com nomes :)

Exercícios - Exemplo de entrada

```
ceci_vieira entrou
dandara entrou
gio_morais entrou
dandara saiu
bia entrou
gio_morais saiu
ceci_vieira saiu
bia saiu
```

Resolução

```
contador = 0
status = input().split()[1] # só status importa
if status == "entrou":
    contador += 1
else:
    contador -= 1

# Continua na próxima página :)
```

Resolução

```
while contador > 0:
    status = input().split()[1] # só status importa
    if status == "entrou":
        contador += 1
    else:
        contador -= 1
    if contador == 24:
        print("Abandonando o Discord e partindo para o Meet!")
        break
else:
    print("Reunião realizada com sucesso!")
```

For... Else

```
for (valor na sequência):
    comandos
else:
    comandos
```

```
for i in range(1, 11):
  if i == 5:
    break
   print(f'Oii, Lady {i}!')
else:
  print(f'Bem vindas!')
```

Exercício

As PyLadies Brasil andam um tanto entediadas na quarentena, por isso resolveram inventar um jogo: **uma delas escolheria um número N e a outra teria C chances de acertar, cada chance que restasse não usada seriam convertidas em pontos para a jogadora, e ela perderá 10 pontos caso não acerte nas C chances.**

Você é a juíza responsável por arbitrar os jogos, e achou por bem fazer um programa para te ajudar com esta tarefa, pois, várias taças de vinho depois, você já não acha a sua memória muito confiável para saber qual era o número que deveria ser encontrado, muito menos quantas chances já foram usadas.

Resolução

```
numero_procurado = int(input().strip())
chances = int(input().strip())
for c in range(chances):
    tentativa = int(input().strip())
    if tentativa == numero_procurado:
        print(f"{chances - c} pontos")
        break
else:
    print("-10 pontos")
```

Para você praticar em casa :)

1. O PyLadies Global realizou uma eleição para compor o conselho. Você foi chamada para escrever um programa para computar os votos. Uma das tarefas é falar qual candidata teve maior quantidade de votos. Assim, faça uma programa que receba os votos de três candidatas e escreva quem foi a candidata vencedora.

   Considere que não houve empate na primeira colocação.

Para você praticar em casa :)

2. O PyLadies Global está realizando uma eleição para adicionar mais uma nova integrante ao conselho. A eleição vai funcionar da seguinte forma:

   Serão 4 candidatas

   Cada capítulo do PyLadies vai votar em uma candidata.

   Escreva um programa que peça o número total de capítulos que irá votar. Em seguida, peça para cada capítulo votar. Ao final da votação, mostre o número de votos de cada candidata.

Para você praticar em casa :)

3. Modifique a questão anterior, para que, além de seu programa exibir a quantidade de votos de cada candidata, ele também informa quem foi a candidata vencedora.

Considere que não houve empate na primeira colocação.

Continue estudando...

Lista de exercícios no HackerRank

Série no Youtube “Introdução a Python: para mulheres que sabem programar em outra linguagem”

e-Book “Introdução a Python: para mulheres que sabem programar em outra linguagem”

Encontro 5

Mulher inspiradora

Liane Tarouco é doutora em Engenharia Elétrica/Sistema Digitais e docente na Universidade Federal do Rio Grande do Sul (UFRGS).
É coordenadora do Programa de Pós-Graduação, e desenvolve pesquisas na área de Informática na Educação sobre mundos virtuais imersivos, aprendizagem experiencial, metodologias ativas e mobile learning.
É a autora do primeiro livro sobre Redes de Computadores no Brasil.

Agenda

- Resolução de exercício
- Lista
- Map
- Enumerate
- Zip

Resolução de Exercício da aula anterior - 1

```
nr_votos_lady1 = int(input("Digite um numero de votos da Lady 1: " ))
nr_votos_lady2 = int(input("Digite um numero de votos da Lady 2: " ))
nr_votos_lady3 = int(input("Digite um numero de votos da Lady 3 : " ))

if (nr_votos_lady1 > nr_votos_lady2)  and (nr_votos_lady1 >
nr_votos_lady3):
   print(f'A Lady 1 venceu a eleição com  {nr_votos_lady1}  votos')
if (nr_votos_lady2 > nr_votos_lady1)  and (nr_votos_lady2 >
nr_votos_lady3):
   print(f'A Lady 2 venceu a eleição com  {nr_votos_lady2}  votos')
if (nr_votos_lady3 >nr_votos_lady1)  and (nr_votos_lady3 > nr_votos_lady2):
   print(f'A Lady 3 venceu a eleição com  {nr_votos_lady3}  votos')
```

Resolução de Exercício da aula anterior - 2

```
numero_capitulos = int(input("Digite a quantidade de capitulos "))
nr_votos_lady1 = nr_votos_lady2 = nr_votos_lady3 = i = 0
while i < numero_capitulos:
   voto = int(input("digite 1 para o lady 1, 2 para o lady 2, 3 para o lady 3 "))
    if voto == 1:
       nr_votos_lady1 = nr_votos_lady1 + 1
   elif voto == 2:
       nr_votos_lady2 = nr_votos_lady2 + 1
   elif voto == 3:
       nr_votos_lady3 = nr_votos_lady3 + 1
   i = i + 1
print(f'Lady 1 teve {nr_votos_lady1} votos')
print(f'Lady 2 teve {nr_votos_lady2} votos')
print(f'Lady 3 teve {nr_votos_lady3} votos')
```

Resolução de Exercício da aula anterior - 3

```
numero_capitulos = int( input("Digite a quantidade de capitulos " ))
nr_votos_lady1 = nr_votos_lady2 = nr_votos_lady3 = i =  0
while i < numero_capitulos:
   voto = int( input("digite 1 para o lady 1, 2 para o lady 2, 3 para o
lady 3 "))
    if voto == 1:
       nr_votos_lady1 = nr_votos_lady1 +  1
   elif voto == 2:
       nr_votos_lady2 = nr_votos_lady2 +  1
   elif voto == 3:
       nr_votos_lady3 = nr_votos_lady3 +  1
   i = i + 1                                  # continua na próxima página
```

Resolução de Exercício da aula anterior - 3

```
print(f'Lady 1 teve {nr_votos_lady1} votos')
print(f'Lady 2 teve {nr_votos_lady2} votos')
print(f'Lady 3 teve {nr_votos_lady3} votos')

if (nr_votos_lady1 > nr_votos_lady2) and (nr_votos_lady1 >
nr_votos_lady3):
   print(f'A Lady 1 venceu a eleição com {nr_votos_lady1} votos')
if (nr_votos_lady2 > nr_votos_lady1) and (nr_votos_lady2 >
nr_votos_lady3):
   print(f'A Lady 2 venceu a eleição com {nr_votos_lady2} votos')
if (nr_votos_lady3 >nr_votos_lady1) and (nr_votos_lady3 > nr_votos_lady2):
   print(f'A Lady 3 venceu a eleição com {nr_votos_lady3} votos')
```

Listas

É uma sequência de valores organizados entre colchetes [ ].

Esses dados podem ser de diferentes tipos: inteiros, floats, strings e inclusive outras listas. Exemplos:

```
dezenas = [10, 20, 30, 40]
comunidades = ['PyLadies Sul de Minas', 'PyLadies Paraíba', 'PyLadies Sorocaba',  ' PyLadies São Carlos',  ' PyLadies Mato Grosso']
lista_vazia = []
```

Listas

Os elementos de uma lista podem ser acessados pelo índice:

```
cores = ['amarelo', 'azul', 'branco', 'dourado']
```

Lembrando que a primeira posição de uma lista em Python é a posição 0:

```
cores[0]
>>> 'amarelo'
cores[2]
>>> 'branco'
```

Listas

Além de ser possível **acessar** cada elemento da lista pelo índice, é possível, **inserir, alterar e deletar**.

Listas são **mutáveis**, ou seja, é possível alterar seus elementos: **adicionando, removendo e/ou substituindo-os.**

Listas

Operador	Explicação	Exemplo	Saída

Concatenação (+)	Cria uma **nova lista com a concatenação das listas passadas**	`natal = ["Débora", "Clara"]`
		`sao_carlos = ["Marília", "Belém"]`
		`ladies = natal + sao_carlos`
			`['Débora', 'Clara', 'Marília', 'Belém']`

Repetição (*)	Cria uma **nova lista com n concatenações da lista passada**	`pyladies = ["PyBar", "Papo", "GTs"]` `vida = pyladies * 3`	`['PyBar', 'Papo', 'GTs', 'PyBar', 'Papo', 'GTs', 'PyBar', 'Papo','GTs']`

Listas

Operador	Explicação	Exemplo	Saída
Indexação `([])`	**Retorna um elemento de uma lista**. O primeiro índice é o zero.	`capitulos = ['Recife', 'São Carlos', 'Salvador', 'Macaé']` `capitulos[1]`	`São Carlos`
`elemento in lista`	Verifica **se um elemento está presente** em uma lista.	`letras = ['P', 'Y', 'L', 'A', 'D', 'I', 'E', 'S']` `'Y' in letras`	`True`
`elemento not in lista`	Verifica **se um elemento está ausente** em uma lista.	`letras = ['P', 'Y', 'L', 'A', 'D', 'I', 'E', 'S']` `'Y' not in letras`	`False`

Listas

Operador	Explicação	Exemplo	Saída

`Fatiamento ([:])`	**Retorna um subconjunto de elementos de uma lista. Final não inclusivo.** Passo é parâmetro opcional que se não passado é 1 por default. Os parâmetros default de início e fim são o início e fim da lista, respectivamente.	`capitulos = ['Recife', 'São Carlos', 'Salvador', 'Macaé', 'Manaus', 'Rio']`
		
		`capitulos[1:3]`
		
		`capitulos[::2]`
		
		`capitulos[1::2]`
			`['São Carlos', 'Salvador']`
			
			`['Recife', 'Salvador', 'Manaus']`
			
			`['São Carlos', 'Macaé', 'Rio']`

Listas

Função	Explicação	Exemplo	Saída
`lista.append(elemento)`	**Adiciona** um novo elemento ao **final da lista**.	`inteiros = [1, 2, 3]` `inteiros.append(10)`	`[1, 2, 3, 10]`
`lista.insert(posicao, elemento)`	**Adiciona** um novo elemento numa **determinada posição da lista**. Todos os elementos daquela posição em diante são deslocados uma posição para frente.	`disciplinas = ["SO", "IA"]` `disciplinas.insert(0, "CAP")`	`['CAP', 'SO', 'IA']`
`lista.remove(elemento)`	**Remove o primeiro elemento especificado encontrado** na lista.	`nomes = ['Ceci', 'Bia']` `nomes.remove('Ceci')`	`['Bia']`

Listas

Função	Explicação	Exemplo	Saída
`lista.`**`count`**`(elemento)`	**Conta quantas vezes o elemento especificado aparece na lista.** Retorna zero se o elemento não pertencer à lista.	`frutas = ["uva", "caju", "uva", "maçã"]` `frutas.count("uva")`	`2`
`lista.`**`index`**`(elemento)`	**Retorna a posição** da primeira ocorrência do elemento especificado. **Caso não seja encontrado, um erro será lançado**.	`tamanhos = ["XP", "M", "GG", "XG"]` `tamanhos.index("GG")`	`2`
`lista.`**`pop`**`(posição)`	**Remove e retorna o elemento da posição especificada.** Caso não seja especificada uma posição, removerá o último elemento da lista.	`inteiros = [5, 6, 7, 8, 9, 10]` `inteiros.pop(2)`	`7`

Listas

Função	Explicação	Exemplo	Saída
`lista.sort()`	Caso os elementos tenham implementação do operador menor (<) e sejam comparáveis (normalmente, de mesmo tipo), Python oferece uma função pronta para **ordenar seu conteúdo.** **Caso os elementos não sejam comparáveis, um erro será lançado.**	`cores = ['rosa', 'verde', 'lilás', 'azul']` `cores.sort()`	`['azul', 'lilás', 'rosa', 'verde']`
`lista.reverse()`	**Inverte as posições dos elementos de uma lista.** O último passa a ser o primeiro, o penúltimo passa a ser o segundo e assim por diante.	`coffee = ['café','bolo', 'refri']` `coffee.reverse()`	`['refri', 'bolo', 'café']`

Listas

Função	Explicação	Exemplo	Saída
`list(objeto_iterável)`	**Cria uma lista a partir de um objeto iterável**, por exemplo, uma string. **Se não for objeto iterável, um erro será lançado**	`palavra = "Python"` `list(palavra)`	`['P', 'y', 't', 'h', 'o', 'n']`
`len(lista)`	Retorna o **tamanho** de uma lista. Funciona para qualquer objeto iterável. **Se não for um objeto iterável, um erro será lançado.**	`ladies = ["Nathália", "Karol", "Juliana"]` `len(ladies)`	`3`
`lista.extend(lista)`	**Insere os elementos da lista** passada por parâmetro **no final da lista original**. É uma concatenação.	`inteiros = [1, 2, 3]` `inteiros.extend([4, 5, 6])`	`[1, 2, 3, 4, 5, 6]`

Listas

Função	Explicação	Exemplo	Saída
`sum(lista_de_numeros)`	Retorna a **soma de uma lista de números.** Só funciona com números.	`inteiros = [16, -40, -15, 38, -4, 25]` `sum(inteiros)`	`20`
`max(lista_ordenavel)`	Retorna o **elemento de valor máximo.** A lista de entrada precisa permitir relações de ordem.	`inteiros = [16, -40, -15, 38, -4, 25]` `max(inteiros)`	`38`
`min(lista_ordenavel)`	Retorna o **elemento de valor mínimo.** A lista de entrada precisa permitir relações de ordem	`inteiros = [16, -40, -15, 38, -4, 25]` `min(inteiros)`	`-40`

Listas

Exemplo de como utilizar algumas das funções apresentadas:

```
coffee = ['refri', 'bolinha de queijo', 'bolo', 'coxinha']
coffee.append('café')
coffee.insert(0, 'água')
coffee.remove('refri')
```

Resultado esperado:

```
['água', 'bolinha de queijo', 'bolo', 'coxinha', 'café']
```

Listas

Para percorrer uma lista toda podemos acessar os elementos diretamente ou utilizar os índices para acessá-los:

```
# acessando diretamente
for elemento in lista
    elemento
# acessando através dos índices
for i in range(len(lista)):
    lista[i]
```

Listas

Atenção!

Acessando **diretamente** você pode apenas percorrer a lista, e **visualizar seu conteúdo**, acessando **pelos índices** você pode, além disso, **alterar o conteúdo de uma posição.**

```
ladies = ["Bia", "Ceci", "Débora"]
ladies[2] = "Débora"
```

A lista resultante será:

```
['Bia', 'Ceci', 'Débora']
```

Map

Map é uma função de ordem superior que **aplica uma função sobre cada elemento de um iterável**. Você pode utilizá-la para aplicar as mais diversas funções, mas inicialmente iremos utilizá-la principalmente **realizar conversões**! Até o momento, você fazia isso:

```
nota1, nota2, nota3 = "9.5 8 10".split()
nota1 = float (nota1)
nota2 = float (nota2)
nota3 = float (nota3)
media = (nota1 + nota2 + nota3) / 3
```

Map

Com map, você pode simplificar as conversões em uma única linha:

```
notas = "9.5 8 10".split()
notas = map(float, notas)
media = sum(notas) / 3
```

Enumerate

Enumerate cria um iterável onde **cada elemento é uma tupla com a posição do elemento original da lista e o próprio elemento**, respectivamente.

```
capitulos = ["PyLadies Natal", "PyLadies Recife"]


for capitulo in enumerate(capitulos): # usando uma única
   print(capitulo)                    # variável de controle


>>(0, 'PyLadies Natal')
>>(1, 'PyLadies Recife')
```

Enumerate

```
capitulos = ["PyLadies Natal", "PyLadies Recife", "PyLadies DF",
"PyLadies São Carlos", "PyLadies Rio"]

for indice, capitulo in enumerate(capitulos):    # com duas
   print(indice, capitulo)                       # variáveis de controle
```

```
0 PyLadies Natal
1 PyLadies Recife
2 PyLadies DF
3 PyLadies São Carlos
4 PyLadies Rio
```

Zip

Zip cria um iterável onde **cada enésimo elemento é uma tupla com os respectivos enésimos elementos das listas utilizadas para gerá-lo.**

Se uma lista for maior que a outra, zip terá o número de elementos da lista menor.

É possível iterar com uma única variável correspondente à tupla criada pelo zip, ou com o número de variáveis correspondente ao número de listas.

Zip

```
capitulos = ["PyLadies Natal", "PyLadies Recife", "PyLadies DF", "PyLadies São Carlos", "PyLadies Rio"]

ladies = ["Débora", "Ana Cecília", "Tânia", "Ana Dulce"]

for pyladies in zip(capitulos, ladies):  # usando uma única
    print(pyladies)                      # variável de controle

>>('PyLadies Natal', 'Débora')
>>('PyLadies Recife', 'Ana Cecília')
>>('PyLadies DF', 'Tânia')
>>('PyLadies São Carlos', 'Ana Dulce')
```

Zip

```
capitulos = ["PyLadies Natal", "PyLadies Recife", "PyLadies DF", "PyLadies São Carlos", "PyLadies Rio"]

ladies = ["Débora", "Ana Cecília", "Tânia", "Ana Dulce"]

for capitulo, lady in zip(capitulos, ladies): # com duas
  print(f"{lady} da {capitulo}") # variáveis de controle

>>Débora da PyLadies Natal
>>Ana Cecília da PyLadies Recife
>>Tânia da PyLadies DF
>>Ana Dulce da PyLadies São Carlos
```

Exercício

Leia um número N e, em seguida, leia N números inteiros. Imprima o maior e o menor entre os N inteiros lidos.

Exemplo de entrada:

```
10
10 -5 1 2 3 100 4 5 -17 1
```

Saída esperada:

```
Maior: 100
Menor: -17
```

Resolução

```
n = int(input())
maior = -1000000
menor = 1000000
entrada = map(int, input().split())
for atual in entrada:
    menor = min(menor, atual)
    maior = max(maior, atual)
print(f"Maior: {maior}")
print(f"Menor: {menor}")
```

Resolução utilizando range

```
n = int(input())
maior = -1000000
menor = 1000000
for i in range(n):
    atual = int(input())
    menor = min(menor, atual)
    maior = max(maior, atual)
print(f"Maior: {maior}")
print(f"Menor: {menor}")
```

Exercício

Você quer **contar quantos canais brasileiros existem no Slack do Pyladies Global, e imprimi-los ordenados alfabeticamente**.

Os canais brasileiros são aqueles cujo nome inicia com “#brasil-”.

Considere que o Slack tem N canais, e que o nome de cada um dos N canais será lido em uma linha.

Resolução

```
num_canais = int(input("Digite o número de canais: "))
canais = []
for i in range(num_canais):
    canal = input("Digite o nome do canal: ")
    if canal.startswith("#brasil-"):
        canais.append(canal)
canais.sort()
print("\n".join(canais))
```

Para você praticar em casa :)

O RH do PyLadies Recife precisa controlar a presença dos membros nas atividades do PyLadies, isto é, precisa anotar na planilha de frequência todos os que participaram.

Como você está estudando Python, se voluntariou a ajudar a tornar essa tarefa um pouco mais fácil, você pretende entregar a lista de presentes já ordenada alfabeticamente para o RH!

Dado um número N, que representa quantas pessoas participaram, e N nomes, imprima os N nomes ordenados alfabeticamente.

Para você praticar em casa :)

Entrada exemplo:

```
4
Teclado
Camila Miho
Ana Carolina
Júlia
```

Saída exemplo:

```
Ana Carolina
Camila Miho
Júlia
Teclado
```

Continue estudando...

Lista de exercícios no HackerRank

Série no Youtube “Introdução a Python: para mulheres que sabem programar em outra linguagem”

e-Book “Introdução a Python: para mulheres que sabem programar em outra linguagem”

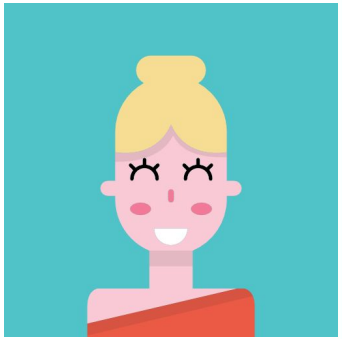
Encontro 6

Mulher inspiradora

Radia Joy Perlman (nascida em 18 de dezembro de 1951) é uma programadora de computadores e engenheira de rede americana. Ela é mais famosa por sua invenção do protocolo de árvore estendida (STP), que é fundamental para a operação de pontes de rede, enquanto trabalhava para a Digital Equipment Corporation. Ela também fez grandes contribuições para muitas outras áreas de design e padronização de rede, como protocolos de roteamento link-state.

Agenda

- Resolução de exercício
- Revisão de “Estrutura de repetição”
- Tuplas
- Dicionários
- Conjuntos



Resolução

```
presentes = int(input())
pessoas = []

for i in range(presentes):
    pessoas.append(input())
pessoas.sort()

print("\n".join(pessoas))
```

Tuplas

Tuplas são **estruturas de dados iteráveis imutáveis**.

Geralmente, têm seus itens delimitados por parênteses , mas também podem ser declaradas simplesmente por **separar os itens com vírgulas**, com exceção de quando se quer criar **uma tupla vazia.**

```
>>> tupla1 = (1,2,3)
>>> tupla2 = 1,2,3
>>> tupla_vazia = ()
>>> print(tupla1, tupla2, tupla_vazia)
(1,2,3) (1,2,3) ()
```

Tuplas

Operador	Explicação	Exemplo	Saída

Concatenação (+)	**Cria uma nova tupla unindo duas tuplas.**	`recife = ("Juliana", "Ana Cecília")`
		`rio = ("Bianca", "Thais")`
		`recife_rio = recife + rio`
			`('Juliana', 'Ana Cecília', 'Bianca', 'Thais')`

Repetição (*)	**Cria uma nova tupla concatenando n repetições da tupla passada.**	`produtoras = ("dands", "gio", "ceci")` `papo_entre_pyladies = produtoras * 3`	`('dands', 'gio', 'ceci', 'dands', 'gio', 'ceci', 'dands', 'gio', 'ceci')`

Tuplas

Operador	Explicação	Exemplo	Saída
Indexação (`[]`)	**Seleciona um elemento de uma tupla.**	`sao_carlos = ("Amanda", "Ana Dulce", "Belém", "Juliana", "Teclado")` `sao_carlos[2]`	`Belém`

Fatiamento (`[:]`)	**Cria uma nova tupla com um subconjunto de elementos de uma tupla**, baseado nos parâmetros inicio, fim e passo, cujos respectivos defaults são zero, tamanho da tupla e um.	`ladies = ("Ana Paula", "Larissa", "Maria Paula", "Pâmela", "Renata")`
		`ladies[1:3]`
		`ladies[2::2]`
			`('Larissa', 'Maria Paula')`
			
			`('Maria Paula', 'Renata')`

Tuplas

Operador	Explicação	Exemplo	Saída
`elemento in tupla`	Verifica **se um elemento está presente** em uma tupla.	`recife = ("Ana Cecília", "Chaina", "Juliana", "Amanda", "Carolina")` `"Chaina" in recife`	`True`
`elemento not in tupla`	Verifica **se um elemento está ausente** em uma tupla.	`produtoras = ("Ana Cecília", "Ana Dulce", "Belém", "Chaina", "Maria Paula", "Alynne", "Maria Paula", "Renata", "Larissa")` `"Maria Paula" not in produtoras`	`False`

Tuplas

Função	Explicação	Exemplo	Saída
`tupla.count(elemento)`	**Conta quantas vezes o elemento especificado aparece na tupla.** Retorna zero se o elemento não pertencer à tupla.	`frutas = "uva", "caju", "uva", "maçã", "açaí"` `frutas.count("uva")`	2
`tupla.index(elemento)`	**Retorna a posição** da primeira ocorrência do elemento passado. **Se não for encontrado, um erro será lançado**.	`tamanhos = ("XP", "M", "GG", "XG")` `tamanhos.index("GG")`	2
`len(lista)`	Retorna o **tamanho** de uma tupla. Funciona para qualquer objeto iterável. **Se não for um objeto iterável, um erro será lançado.**	`ladies = ("Renata", "Maria Paula", "Belém")` `len(ladies)`	3

Informação extra sobre tuplas

ATENÇÃO: tuplas podem conter objetos mutáveis, como listas.

Ao inserir um novo número na lista dentro da tupla (posição 0), não alteramos em nada a tupla, já que as referências que ela guarda são as mesmas! Apenas modificamos a lista dentro dela, **mas sem mudar o identificador.**

```
>>> tupla = ([1,2,3],[4,5])
>>> id(tupla)
140393279659016
>>> tupla[0].append(4)
>>> tupla
([1,2,3,4], [4,5])
>>> id(tupla)
140393279659016
```

Tuplas

Geralmente, vemos as **listas** possuindo itens todos do mesmo tipo, enquanto tuplas são mais encontradas obtendo itens de diversos tipos, **mas isso não significa que listas não possam ser heterogêneas e tuplas homogêneas!**

> “As listas são destinadas a serem sequências homogêneas, enquanto que os Tuplas são estruturas de dados heterogêneas.”
>
> Phillip J. Eby

Exercício

Crie duas tuplas: uma contendo as suas 3 frutas preferidas e outra contendo suas 3 cores preferidas.

Imprima na tela os itens das duas tuplas combinadas elemento por elemento separadas por dois pontos.

Dica: lembre-se de utilizar f-strings para a formatação da saída.

Exercício

Entrada:

```
frutas = ("pêssego", "morango", "kiwi")
cores = ("violeta", "azul", "verde")
```

Saída esperada:

```
pêssego: violeta
morango: azul
kiwi: verde
```

Resolução

```
frutas = ("pêssego", "morango", "kiwi")
cores = ("violeta", "azul", "verde")

for i, j in zip(frutas, cores):
    print(f"{i}: {j}")
```

Dicionários

Dicionário é um objeto que **relaciona uma chave a um valor**.

A **chave** é o que **indexa** um dicionário, isto é, ela corresponde ao conceito de índice.

Desta forma, em vez de ter posições 0, 1, 2, 3 e assim por diante, podemos ter posições com os valores que desejamos, como “azul”, “amarelo” e “vermelho”.

Por se tratarem de índices, uma consequência direta é que as **chaves de um dicionários precisam ser únicas**, e, caso você declare mais de uma vez a mesma chave, o valor será **sobreposto**.

Dicionários

```
dicionario_vazio = {}
camiseta = {"M":12, "G": 10, "GG":10}
dados = {
    "grupo": "PyLadies São Carlos",
    "fundação":2014,
    "integrantes": 37
}
alunas_curso = {"Ana": "BCC", "Amanda": "EnC"}
alunas_curso["Belém"] = "Biotec"
```

Dicionários

Para **criar um dicionário vazio**, atribua abre e fecha chaves à variável:

```
dicionario = {}
```

Para **adicionar uma nova chave e seu respectivo valor** a um dicionário existente, ou **atualizar um valor de uma chave existente**, atribua o novo valor ao dicionário na "posição" da nova chave:

```
dicionario['chave'] = 'valor'
```

Para **acessar um valor, deve-se passar a chave:**

```
dicionario['chave']
```

Dicionários

Operador	Explicação	Exemplo	Saída
`chave in dicionario`	Verifica **se uma chave existe** em um dicionário.	`camisetas_dg = {"P": 3, "M": 1}` `"M" in camisetas_dg`	`True`
`chave not in dicionario`	Verifica **se uma chave não existe** em um dicionário.	`camisetas_dg = {"P": 3, "M": 1}` `"M" not in camisetas_dg`	`False`
`del (dicionario [chave])`	**Deleta uma chave e seu respectivo valor** de um dicionário.	`fundacao = {2013: ["Natal"], 2014: ["Distrito Federal", "Recife", "São Carlos"], 2021:[]}` `del (fundacao[2014])`	`{2013: ['Natal'], 2021: []}`

Dicionários

Função	Explicação	Exemplo	Saída

`dicionario.` **`keys`**`()`	Retorna um objeto iterável com **todas as chaves** de um dicionário.	`brindes = {"caneca": 100, "bottom": 500 , "stickers": 1000}`
		`for brinde in brindes.keys():`
		`  print(brinde)`
			`caneca`
			`bottom`
			`stickers`

`dicionario.` **`values`**`()`	Retorna um objeto iterável com **todos os valores** de um dicionário.	`brindes = {"caneca": 100, "bottom": 500 , "stickers": 1000}`
		`for qtd in brindes.values():`
		`  print(qtd)`
			`100`
			`500`
			`1000`

Dicionários

Função	Explicação	Exemplo	Saída

`dicionario.items()`	Retorna um objeto iterável com **um tupla contendo chave e valor** de um dicionário. Pode ser iterado com uma variável, que assumirá o valor da tupla inteira, ou duas, onde chave e valor terão variáveis respectivas.	`brindes = {"caneca": 100, "bottom": 500 , "stickers": 1000}`
		`for item in brindes.items():`
		`  print(item)`
		
		`for brinde, qtd in brindes.items():`
		`  print(f"{qtd} unidades de {brinde}")`
			`('caneca', 100)`
			`('bottom', 500)`
			`('stickers', 1000)`
			`100 unidades de caneca`
			`500 unidades de bottom`
			`1000 unidades de stickers`

Dicionários

Função	Explicação	Exemplo	Saída
`dicionario.copy()`	**Retorna um novo dicionário, com uma cópia real do objeto.** Se você fizer dict1 = dict2, ele criará apenas uma referência.	`brindes = {"caneca": 100, "bottom": 500 , "stickers": 1000}` `campanha = brindes.copy()`	`{'bottom': 500, 'caneca': 100, 'stickers': 1000}`
`dicionario.clear()`	**Deleta todo o conteúdo** do dicionário e retorna um dicionário vazio.	`alunas = {"Introdução à Programação com Python": 60, "Introduçãão a Python": 57, "Python Intermediário": 79}` `alunas.clear()`	`{}`

Dicionários

Função	Explicação	Exemplo	Saída
`dicionario.` **`get`**`(chave)`	**Retorna o valor de uma dada chave, ou None caso ela não exista**. É aconselhável o uso em vez de acesso direto em casos onde uma chave possa não existir, pois no acesso direto causaria erro.	`coffee = {"bolo": 10, "refri": 6, "café": 3, "coxinha": 300, "pão de queijo": 300}` `coffee.get("pão de queijo")`	`300`
`dicionario.` **`pop`**`(chave)`	**Retorna o valor de uma dada chave e exclui todo o elemento do dicionário.**	`camisetas_dg = {"P": 3, "M": 1}` `camisetas_dg.pop("P")`	`3`

Dicionários

Função	Explicação	Exemplo	Saída
`dicionario.` **`popitem`**`()`	Retorna um **elemento do dicionário e o exclui do dicionário.** Não é possível escolher qual elemento será retornado.	`camisetas_dg = {"P": 3, "M": 1}` `camisetas_dg.popitem()`	`('M', 1)`
`dicionario.` **`update`**`(` `dicionario)`	**Atualiza elementos existentes ou adiciona elementos** caso estes não existam num dicionário base.	`camisetas_dg = {"P": 3, "M": 1}` `camisetas_dg.update({"G": 3, "GG": 2})`	`{'G': 3, 'GG': 2, 'M': 1, 'P': 3}`

Dicionários

Para **iterar** sobre dicionários usando a **chave** como acesso:

```
for chave in dicionario:
    dicionario[chave]
```

Para **iterar** sobre os conjuntos **chave valor** de uma vez:

```
for item in dicionario.items():
    item # a saída é uma tupla, com elementos chave e valor
for chave, valor in dicionario.items():
    chave, valor # ou chave e valor explicitamente
```

Dicionários

```
universidade = {}
universidade['USP'] = 'Amarelo'
universidade['UFSCar'] = 'Vermelho'
'USP' in universidade
>>>True
'UNESP' in universidade
>>>False
'Amarelo' in universidade
>>>False
del universidade['USP']
universidade
>>> {'UFSCar': 'Vermelho'}
```

Exercício

As PyLadies estão votando qual será a cor da identidade visual da PyLadies Conf deste ano, e você ficou responsável por processar a lista completa de votos e exibir o resultado.

Será fornecido um número N de votos, seguidos por N linhas, cada uma com uma cor.

Seu trabalho é **agrupar todas as cores e determinar qual teve mais votos**. Você não deve distinguir letras maiúsculas de minúsculas pois "Azul" e "azul" obviamente representam a mesma cor. Trate também espaços em branco no início e final das linhas.

Resolução

```
n = int(input("Digite o número de votos: "))
votos = {}
for i in range(n):
    cor = input("Digite a cor: ").strip().lower()
    if cor in votos:
        votos[cor] += 1
    else:
        votos[cor] = 1
print(f"A cor mais votada é {max(votos, key =
votos.get)}!")
```

Exercício

Dada uma palavra, retorne um dicionário cuja chave é a posição de cada letra (iniciando em 1) e o valor é a própria letra.

Entrada exemplo:

```
'I<3pY'
```

Saída exemplo:

```
{1: 'I', 2: '<', 3: '3', 4: 'P', 5: 'y'}
```

Resolução

```
palavra = input ("Digite uma palavra: ")
dicionario = {}

for i, letra in enumerate(palavra):
    dicionario[i+1] = letra

print(dicionario)
```

Conjuntos

(int, string ou qualquer objeto cuja classe implemente o método __hash__)

Segundo a documentação oficial de Python, um conjunto é:

"Um objeto set é uma coleção não ordenada de objetos hasheáveis distintos."

Em Python, um conjunto é representado por **valores entre chaves**.

OBS: cuidado para não confundir com dicionários.

Conjuntos

Declarando um conjunto vazio:

```
conjunto_vazio = set()
```

Criando um conjunto a partir de uma lista:

```
novo_conjunto = set(lista)
```

Criando um conjunto com valores:

```
novo_conjunto = {1, 2, 3, 4, 5}
```

Conjuntos

Operador	Explicação	Exemplo	Saída
`elemento in conjunto`	Verifica se um elemento **pertence** ao conjunto.	`locais = {"Goiânia", "Nova Iguaçu", "Sul de Minas", "Sergipe"}` `"Sergipe" in locais`	`True`
`elemento not in conjunto`	Verifica se um elemento **não pertence** ao conjunto.	`locais = {"Goiânia", "Nova Iguaçu", "Sul de Minas", "Sergipe"}` `"Goiânia" not in locais`	`False`
`len(conjunto)`	Retorna o **tamanho** do conjunto	`locais = {"Goiânia", "Nova Iguaçu", "Sul de Minas", "Sergipe"}` `len(locais)`	`4`

Conjuntos

Função	Explicação	Exemplo	Saída
`conjunto.` **`add`**`(elemento)`	**Adiciona um elemento a um conjunto.**	`locais = {"Parnaíba", "Rosana", "Blumenau"}` `locais.add("Macaé")`	`{'Blumenau', 'Macaé', 'Parnaíba', 'Rosana'}`
`conjunto.` **`remove`**`(elemento)`	**Remove um elemento especificado de um conjunto e retorna None.** Se não houver o elemento, um erro será lançado	`locais = {'Blumenau', 'Macaé', 'Parnaíba', 'Rosana'}` `locais.remove('Macaé')`	`{'Blumenau', 'Parnaíba', 'Rosana'}`

Conjuntos

Função	Explicação	Exemplo	Saída
`conjunto.`**`pop`**`()`	**Retorna e exclui um elemento do conjunto.** Não se determina o elemento escolhido, o algoritmo escolhe.	`locais = {'Blumenau', 'Macaé', 'Parnaíba'}` `locais.pop()`	`'Blumenau'`
`conjunto.`**`clear`**`()`	**Remove todos os elementos** de um conjunto.	`locais = {'Blumenau', 'Macaé', 'Parnaíba'}` `locais.clear()`	`{}`
`conjunto.`**`update`**`(conjunto)`	**Adiciona os elementos** do conjunto passado no conjunto original.	`locais = {'Blumenau', 'Macaé'}` `locais.update('Rosana')`	`{'Blumenau', 'Macaé', 'Parnaíba'}`

Conjuntos

Função	Explicação	Exemplo	Saída

`conjunto.`**`issubset`**`(conjunto)`	Retorna verdadeiro se o conjunto **está contido** no conjunto passado.	`brasil = {"Paraíba", "Caxias do Sul", "Florianópolis", "São José dos Campos", "Manaus", "Sorocaba"}`
		`sp = {"Sorocaba", "São José dos Campos"}`
		`sp.issubset(brasil)`
			`True`

`conjunto.`**`issuperset`**`(conjunto)`	Retorna verdadeiro se o conjunto **contém** o conjunto passado.	`brasil = {"Paraíba", "Caxias do Sul", "Florianópolis", "São José dos Campos", "Manaus", "Sorocaba"}`
		`sp = {"Sorocaba", "São José dos Campos"}`
		`brasil.issuperset(sp)`
			`True`

Conjuntos

Função	Explicação	Exemplo	Saída

`conjunto.`**`union`**`(conjunto)`	**Retorna um novo conjunto** com a união dos elementos dos dois conjuntos.	`brasil = {"Maceió", "Ribeirão Preto", "Teresina"}`
		`rs = {"Porto Alegre"}`
		`rs.union(brasil)`
			`{"Maceió", "Porto Alegre", "Ribeirão Preto", "Teresina"}`

`conjunto.`**`isdisjoint`**`(conjunto)`	**Retorna verdadeiro se os conjuntos são disjuntos**, isto é, se eles não tem nenhum elemento em comum.	`brasil = {"Maceió", "Ribeirão Preto", "Teresina"}`
		`rs = {"Porto Alegre"}`
		`rs.isdisjoint(brasil)`
			`True`

Conjuntos

Função	Explicação	Exemplo	Saída

`conjunto.` **`intersection`** `(conjunto)`	**Retorna um novo conjunto com os elementos comuns** entre o primeiro e segundo conjunto. É a operação inversa do symmetric difference.	`multiplos_6 = {12, 18, 24}`
		`multiplos_4 = {8, 12, 16, 20, 24}`
		`multiplos_6.intersection(multiplos_4)`
			`{12,24}`

`conjunto.` **`intersection_update(`** `conjunto)`	**Retira do primeiro conjunto os elementos que não estão na intersecção** entre os dois conjuntos. É a operação inversa do symmetric difference update.	`multiplos_6 = {12, 18, 24}`
		`multiplos_4 = {8, 12, 16, 20, 24}`
		`multiplos_6.intersection_update(multiplos_4)`
			`{12,24}`

Conjuntos

Função	Explicação	Exemplo	Saída

`conjunto.` **`difference`** `(conjunto)`	**Retorna um novo conjunto com os elementos do primeiro conjunto que não estão contidos no segundo.**	`multiplos_6 = {12, 18, 24}`
		`multiplos_4 = {8, 12, 16, 20, 24}`
		`multiplos_6.difference(multiplos_4)`
			`{18}`

`conjunto.` **`difference_update`** `(conjunto)`	**Retira os elementos da intersecção dos dois conjuntos do primeiro conjunto.**	`multiplos_6 = {12, 18, 24}`
		`multiplos_4 = {8, 12, 16, 20, 24}`
		`multiplos_6.difference_update(multiplos_4)`
			`{18}`

Conjuntos

Função	Explicação	Exemplo	Saída

`conjunto.` **`difference`** `(conjunto)`	**Retorna um novo conjunto com os elementos únicos dos dois conjuntos.** É a operação inversa do intersection.	`multiplos_6 = {12, 18, 24}`
		`multiplos_4 = {8, 12, 16, 20, 24}`
		`multiplos_6.symmetric_difference(multiplos_4)`
			`{8, 16, 18, 20}`

`conjunto.` **`symmetric_difference_update`** `(conjunto)`	**Atualiza o primeiro conjunto com os elementos únicos dos dois conjunto**. É a operação inversa do intersection_update.	`multiplos_6 = {12, 18, 24}`
		`multiplos_4 = {8, 12, 16, 20, 24}`
		`multiplos_6.symmetric_difference_update(multiplos_4)`
			`{8, 16, 18, 20}`

Exercício

As PyLadies de São Carlos estão organizando uma votação para a cor da nova camiseta. Você foi o encarregada de pegar as opiniões do grupo e tirar as cores duplicadas.

Cada cor é **composta por letras de a a z minúsculas**, podem conter hifens e sem espaço. Insira todas as cores coletadas em uma única linha e separadas apenas por espaços.

Imprima os nomes das cores ordenados alfabeticamente.

Resolução

```
cores = set(input().split())
print('\n'.join(sorted(cores)))
```

Para você praticar em casa :)

As PyLadies estão dividindo as integrantes entre grupos de trabalho na quarentena, mas o RH está preocupado com a saúde mental das pessoas. Por isso, você ficou responsável por fazer um programa que cheque se existe alguém participando de mais de um grupo de trabalho.

Leia um número G que representa quantos grupos de trabalho existem. Para cada grupo, leia seu nome, um inteiro N que representa o número de membros daquele grupo, e, em seguida, N nomes, um em cada linha. Informe “OK” se não houver pessoas repetidas, e “Reveja as distribuições” caso contrário.

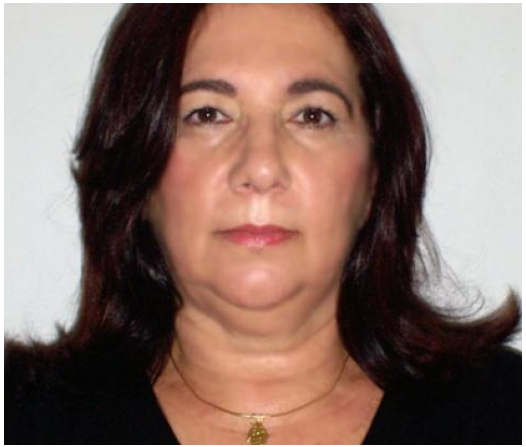
Continue estudando...

- e-Book “Introdução a Python: para mulheres que sabem programar em outra linguagem”

- Lista de exercícios no HackerRank

Encontro 7

Mulher inspiradora



Judith Kelner é professora titular do Centro de Informática da UFPE, tem doutorado em Computer Science pela Universidade de Kent (Inglaterra) e seus interesses de pesquisa são Visão Computacional, Visualização de Dados, Interfaces Avançadas e Internet das Coisas. Ela é uma das fundadoras do Grupo de Pesquisa em Realidade Virtual e Multimídia (GRVM), referência nacional em visão computacional, e atualmente lidera uma equipe de 26 pesquisadores desse grupo na criação de soluções inovativas para problemas complexos.

Agenda

- Resolução de exercício
- Funções
- Classes

Resolução

```
num_grupos = int(input("Digite o número de grupos: "))
grupos = {}
for i in range(num_grupos):
    grupo = input("Digite o nome do grupo: ")
    grupos[grupo] = []
    num_integrantes = int(input("Digite o número de integrantes:"))
    for integrante in range(num_integrantes):
        grupos[grupo].append(input("Digite o nome da integrante: "))
# Continua na próxima página
```

Resolução

```
total = set()
for nome, integrantes in grupos.items():
    if total.isdisjoint(set(integrantes)):
        total.update(set(integrantes))
    else:
        print("Reveja as distribuições!")
        break
else:
    print("OK")
```

Função

Funções em Python são declaradas com a palavra reservada **def**, possuem **argumentos** opcionais e podem ou não **retornar algum valor**.

```
def funcao(argumentos_opcionais):
    codigo
    return opcional
```

Exercício

O PyLadies Brasil tem um grupo de trabalho focado na criação de materiais, e você foi chamada para ajudá-lo!

Faça uma função que receba o nome de uma Lady e a região de onde é seu capítulo e, de acordo com a região, mostre a seguinte mensagem:

"Olá [nome-da-Lady], o grupo precisa que você ajude na aula [número-aula]!"

Região	Aula
Centro-Oeste	1
Nordeste	2
Norte	3
Sudeste	4
Sul	5

Resolução

```
def criacao_mat(nome, regiao):
    distribuicao = {"Centro-Oeste": 1, "Nordeste": 2,
"Norte": 3, "Sudeste": 4, "Sul": 5}

    print(f"Olá {nome}, o grupo precisa que você ajude com
a aula {distribuicao[regiao]}!")

criacao_mat('Amanda', 'Norte')
```

Classes

Classes são criadas utilizando a palavra reservada **class**, tem métodos definidos com a palavra reservada **def**.

Atenção à indentação e ao self! Para um método pertencer à uma classe, ele precisa estar indentado dentro da classe e definido com o argumento self.

Os métodos funcionam de forma análoga às funções, possuindo parâmetros e retorno opcionais.

Classes

```
class NomeDaClasse (HerancaOpcional):
    # Se não herdar de outra classe, não precisa nem dos parênteses
    def __init__(self, argumentos_opcionais):
        # Inicializador da classe
        # É executado quando a classe é instanciada

    def metodos_da_classe(self, argumentos_opcionais):
        # Realizam operações sobre a classe
```

Classes

```
class PyLadies:
    # Inicializador da classe
    def __init__(self, nome_capitulo, num_integrantes,
integrantes):
        self.nome_capitulo = nome_capitulo
        self.num_integrantes = num_integrantes
        self.integrantes = integrantes
        print('Capítulo criado!')
# Continua na próxima página
```

Classes

```
    def __str__(self):
        return f"PyLadies
{self.nome_capitulo}\n{self.num_integrantes}
integrantes:\n" + "\n".join(self.integrantes)

    def adicionar_integrante(self, nome):
        self.num_integrantes += 1
        self.integrantes.append(nome)
# Métodos da classe - Continua na próxima página
```

Classes

```
# Instanciando e inicializando a classe
pyladies_sao_carlos = PyLadies('São Carlos', 6, ["Amanda",
"Ana Dulce", "Hemilyn", "Juliana Karoline", "Juliana
Alves", "Thayana"])
# Chamando o método adicionar integrante
pyladies_sao_carlos.adicionar_integrante("Marília")
# Utilizando o método especial __str__ para exibir a
classe
print(pyladies_sao_carlos)
```

Classes

Em Python, o **inicializador** da classe é o **__init__**, que faz parte dos conhecidos ***dunder methods***, os “métodos mágicos” do Python.

Existem muitos dunder methods que não iremos abordar neste momento, focaremos em __init__, que inicializa a classe, e __str__, que cria uma representação da classe voltada ao usuário e permite que um objeto seja passado diretamente em um print.

Exercício

Crie mais um método para a classe PyLadies definida anteriormente para ordenar as integrantes alfabeticamente.

Então, crie dois objetos da classe PyLadies, com nome, ano de fundação e integrantes, utilize seu novo método para ordenar os nomes das integrantes, e exiba o objeto em um print.

Resolução

```
    # Método indentado dentro da classe
    def ordenar_integrantes(self):
        self.integrantes.sort()


# Instanciando a classe PyLadies
pyladies_natal = PyLadies("Natal", 2, ["Débora", "Clara"])
pyladies_natal.ordenar_integrantes()
print(pyladies_natal)
# Continua na próxima página
```

Resolução

```
# Instanciando a classe PyLadies
pyladies_recife = PyLadies("Recife", 3, ['Chaina', 'Ana Cecília', 'Juliana'])
pyladies_recife.ordenar_integrantes()
print(pyladies_recife)
```

Continue estudando...

Lista de exercícios no HackerRank

Série no Youtube “Introdução a Python: para mulheres que sabem programar em outra linguagem”

e-Book “Introdução a Python: para mulheres que sabem programar em outra linguagem”

Referências

PEP - https://www.python.org/dev/peps. Acessado em: set. 2020

PEP 8 (Style guide for python code) - https://www.python.org/dev/peps/pep-0008/. Acessado em: set. 2020

Material PyLadies: São Carlos - https://docs.google.com/presentation/d/1CLawydiOsuAwORx3Pu07hpWIYL-1trhTBuotu8HxAIM/edit#slide=id.g55b48b4b4b80_0_79

Material PyLadies: São Paulo -

1 - Alana Morais, Aline Morais. PROTAGONISMO FEMININO NA COMPUTAÇÃO: Desmistificando a Ausência de Mulheres Influentes na Área Tecnológica. Cabedelo - PB: Editora Unesp. 2019. Disponível em: http://editora.iesp.edu.br/index.php/UNIESP/catalog/book/90

2 - Allen B. Downey. Pense Python. Editora Novatec. trad. Disponível em: https://penseallen.github.io/PensePython2e/

3 - Documentação oficial. Disponível em: https://docs.python.org/pt-br/3/contents.html