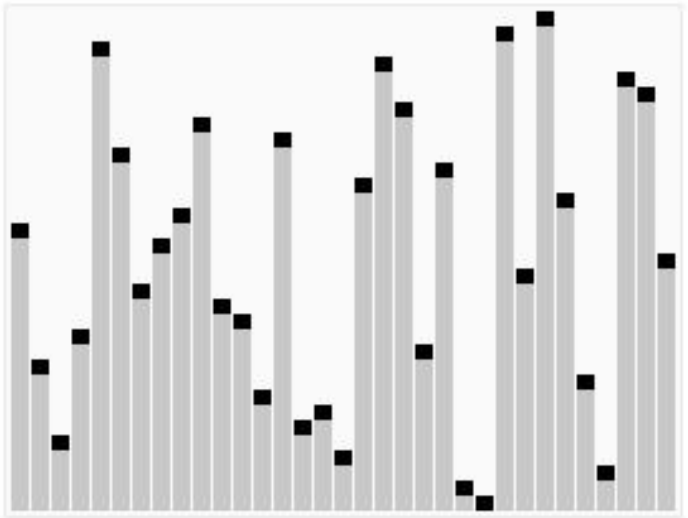
Introdução a lógica com Python

prof. Rodrigo Attique

Definição

Em matemática e ciência da computação, um **algoritmo** é uma sequência finita de ações executáveis que visam obter uma solução para um determinado ti



Onde usar algoritmos?

Estão presentes em quase tudo na vida real;

Manuais de instruções

Tutoriais

ETC

Então….

Qualquer passo a passo que eu crie para resolver um problema é um algoritmo.

Algoritmo trocar lâmpada

1. Desligue a eletricidade;
2. Posicione a escada sob a lâmpada;
3. Abra a escada;
4. Suba na escada até chegar na lâmpada;
5. Remova a lâmpada antiga;
6. Desça da escada;
7. Descarte a lâmpada velha;
8. Pegue a lâmpada nova;
9. Suba na escada com ela;
10. Encaixe a nova lâmpada;

1. Desça da escada;
2. Guarde a escada;
3. Acenda a luz;

Desafios!

1. Faça um algoritmo para trocar um bebê.
2. Faça um algoritmo para trocar o pneu de um carro.

Trocar um bebê

1. Coloque o bebê sobre o trocador;
2. Retire a fralda suja;
3. Descarte a fralda suja;
4. Limpe o bebê com lenço umedecido;
5. Passe talco nas partes;
6. Coloque a nova fralda;

Trocar pneu do veículo

1. Estacione o carro num local seguro;
2. Acione o pisca alerta;
3. Pegue a chave de rodas, step e triângulo;
4. Coloque o triângulo a 50m da traseira do veículo;
5. Com a chave de rodas afrouxe os parafusos usando o peso do corpo;
6. Posicione o macaco na lateral do veículo proximo da roda furada;
7. levante o carro até o pneu sair um pouco do chão.
8. Remova os parafusos;
9. Remova pneu furado;

1. Colocque novo pneu
2. Coloque os parafusos
3. Desça o carro;
4. Remova o macaco
5. Aperte os parafusos com a chave de rodas usando o peso do corpo;
6. guarde o macaco, chave de rodas trinagulo e pneu furado;

Mas antes um pouco de portugol não faz mal

- O portugol ou pseudo-código é usado para esboçar um programa antes de escrevê-lo em uma linguagem de programação.
- Assim o analista pode escrever um algoritmo usando palavras da sua língua natural antes de programar definitivamente.

Análise léxica ou palavras reservadas

Toda linguagem possui o que se chama de lexemas ou palavras reservadas.

Dessa forma antes de compilar/interpretar (processo que traduz o código para linguagem de máquina).

leia, escreva, inteiro, flutuante…

Análise sintática

Depois o compilador procura pela sintaxe do código.

De forma que o programador tenta empregado as palavras reservadas na ordem e forma correta;

inteiro x

escreva(“Seu nome”)

leia(x)

Estrutura de um programa

Importações

Declaração de Variáveis

Processamento

Saídas

Como em uma redação um programa precisa de começo, meio e fim.

Importações: Sempre no topo do programa, são trechos de código prontos que podemos trazer pro código quando necessário.

Declaração de Variáveis: São pequenos registradores de valor, que devem ser inicializados para o processamento.

Estrutura de um programa

Importações

Declaração de Variáveis

Processamento

Saídas

Como em uma redação um programa precisa de começo, meio e fim.

Processamento: Depois de importar e declarar nossas variáveis precisamos processar todos os dados.

Saídas: Por fim mostrar ao usuário os resultados obtidos com todos os dados coletados ou importados.

Estrutura de um programa

```
# Importações

# Declaração de Variáveis

largura = 10 # largura recebe 10

altura = 9 # altura recebe 9

area = 0 # area recebe 0

# Processamento

area = largura * altura

# Saídas

imprima(“Area total = ”, area)
```

Como em uma redação um programa precisa de começo, meio e fim.

Processamento: Depois de importar e declarar nossas variáveis precisamos processar todos os dados.

Saídas: Por fim mostrar ao usuário os resultados obtidos com todos os dados coletados ou importados.

Atribuindo valores em variáveis

No programa acima nós declaramos 3 variáveis, largura, altura e area.

Sabemos que são variáveis por estão sempre á esquerda de um atribuidor ‘=’ (sinal de igual).

Sabemos também que são tipos de números inteiros por causa dos valores que foram atribuídos.

Comentários

- Comentários são essenciais em todo o código.
- Cada linguagem tem uma forma de escrever comentários.
- Aqui estamos dizendo que tudo que é escrito depois de # (cerquilha) é um comentário.
- Por tanto não será compilado no programa.

Nomes de variáveis

É importante ressaltar que a máquina precisa entender o que foi escrito, há palavras que o usuário criará no programa, a exemplo das variáveis.

Por tanto vamos escrevê-las sempre:

- sem acentos, cedilhas (ç).
- começando com letras minúsculas
- quando for mais de uma palavra usar um sublinha (_)

Nomes de variáveis - Exemplo

- **Correto:**

largura = 10

batatinha_frita_123 = 9

nome10 = “Fulano”

- **Errado:**

Largura = 10

batatinha frita 123 = 9

10nome = “Fulano”

Tipos básicos de dados

- Na matemática temos os conjuntos numéricos básicos dos números
- **reais** (com vírgula) que chamamos de flutuante e dentro deles temos os número **inteiros** (sem vírgula)
- Na programação além desse tipos temos o tipo **cadeia**, usado para escrever palavras.
- E o tipo **booleano** (verdadeiro, falso)

Tipos básicos de dados

- Dessa forma podemos ter variáveis…

a = 10 # a recebe um inteiro 10

b = 10.2 # b recebe o flutuante 10.2

c = “fulano” # c recebe uma cadeia fulano

d = Verdadeiro # d recebe um booleano True

Vamos praticar com o python

Você pode instalar o python por este link.

Não se esqueça de adicionar o python ao path marcando a segunda opção logo no início.

Estrutura de um programa

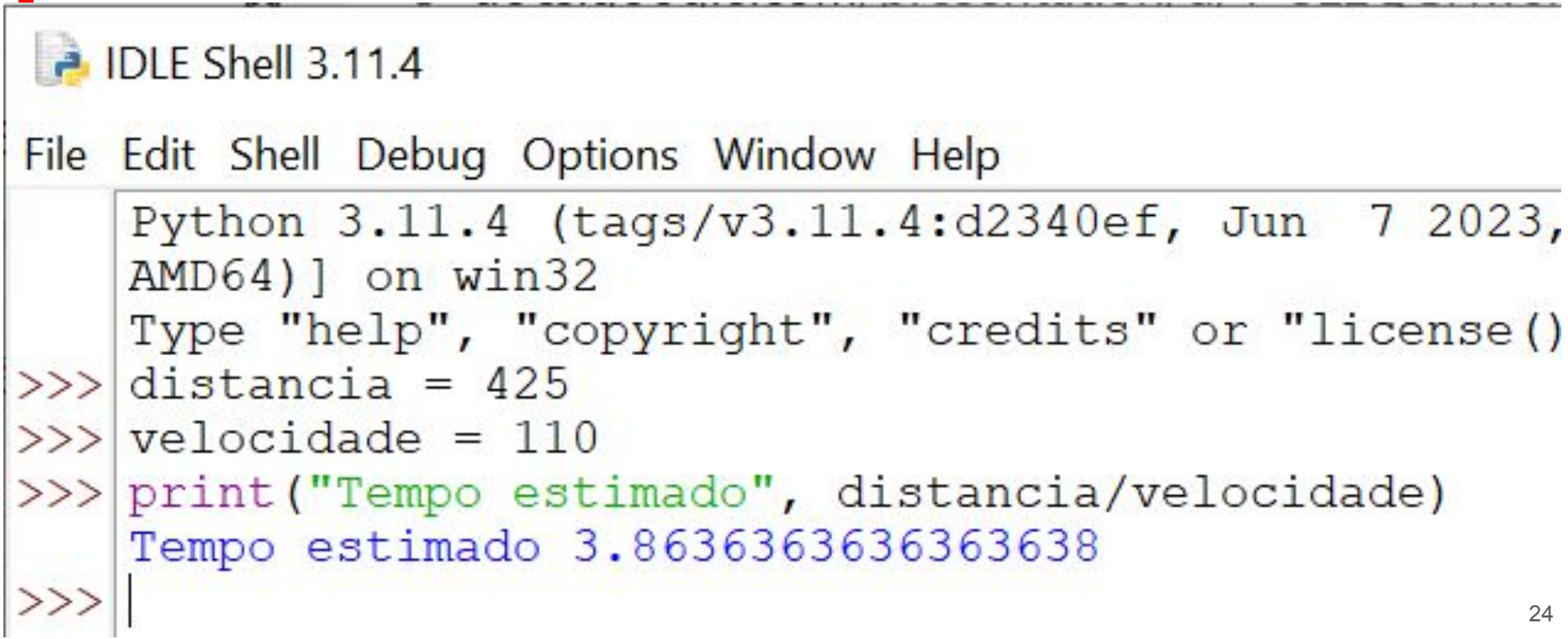
```
# Importações

# Declaração de Variáveis

largura = 10 # largura recebe 10

altura = 9 # altura recebe 9

area = 0 # area recebe 0

# Processamento

area = largura * altura

# Saídas

print(“Area total = ”, area)
```

- **Nosso programa em python segue a mesma estrutura do anterior.**
- **Porém usando palavras do inglês.**

Área total

```
Type "help", "copyright", "credi
>>> #programa área
>>> comprimento = 5
>>> distancia = 10
>>> area = comprimento * distancia
>>> print("Area total é", area)
Area total é 50
```

Algoritmo de distância



IDLE

O python possui um shell interativo onde é possível digitar comandos em tempo de execução.

Ele é útil para testarmos código antes de inserir no código principal ou para resolver problemas simples.

Variáveis

- São dados armazenados na memória RAM do computador;
- velocidade = 110 #Lê-se velocidade recebe 110
- Uma variável sempre deve estar a esquerda do valor recebido ou seja a esquerda do sinal de igual (atribuidor).
- **Toda variável deve ser declarada antes de qualquer lógica do programa**

Quer dizer…

Quer dizer que toda variável declarada no programa reserva um pequeno espaço na memória ram para aquele valor.

Logo a memória RAM é como um condomínio que disponibiliza espaço para locação temporária de dados.

Operadores matemáticos

Saiba mais

+	Soma
-	Subtração
/	Divisão
*	Multiplicação
%	Resto
**	Potenciação
//	Divisão Inteira

Como utilizar os operadores matemáticos em Python?

Para utilizar os operadores matemáticos em Python, é necessário conhecer a sintaxe correta e entender como aplicá-los em expressões e variáveis. Vamos ver alguns exemplos práticos de como utilizar cada um dos operadores matemáticos em Python:

1. Operador de adição (+):

```
1    a = 5
2    b = 3
3    resultado = a + b
4    print(resultado)  # Output: 8
5    nome1 = "João"
6    nome2 = "Maria"
7    nome_completo = nome1 + " " + nome2
8
```

2. Operador de subtração (-):

```
1    a = 10
2    b = 7
3    resultado = a - b
4    print(resultado)  # Output: 3
```

3. Operador de multiplicação (*):

```
1    a = 4
2    b = 5
3    resultado = a * b
4    print(resultado)  # Output: 20
```

4. Operador de divisão (/):

```
1    a = 10
2    b = 2
3    resultado = a / b
4    print(resultado)  # Output: 5.0
```

5. Operador de potenciação (**):

```
1    a = 2
2    b = 3
3    resultado = a ** b
4    print(resultado)  # Output: 8
```

6. Operador de resto da divisão (%):

```
1    a = 10
2    b = 3
3    resultado = a % b
4    print(resultado)  # Output: 1
```

7. Operador de divisão inteira (//):

```
1    a = 10
2    b = 3
3    resultado = a // b
4    print(resultado)  # Output: 3
```

8. Operador de negação (-):

```
1    a = 5
2    resultado = -a
3    print(resultado)  # Output: -5
```

Formatando nossas mensagens

Usar a função print sem um formatador é contraproducente, por isso vamos usar um formatador.

Adicione um ‘**f**’ antes de iniciar a string, assim podemos passar nossas variáveis dentro de chaves.

```
>> distancia = 10
>> print(f"Á área informdada é {area}")
   Á área informdada é 50
```

Tente fazer…

nome = “Coloque seu nome aqui…”

idade = 9 # substitua pela sua idade

mensagem = f”Olá, {nome}, parece que você tem {idade} anos.”

print(mensagem)

Funções embutidas

Todas linguagem possui uma função embutida.

Sabemos que é uma função por causa do parênteses ao final dela.

Exemplo:

print(), input()...

Pegando dados do usuário

É possível interagir com o usuário armazenando os dados que ele digita usando a função input.

```
> nome = input("Seu nome:" )
  Seu nome:Rodrigo
```

Abandonando o IDLE

O IDLE tem seu charme mas é preciso evoluir para algo mais produtivo.

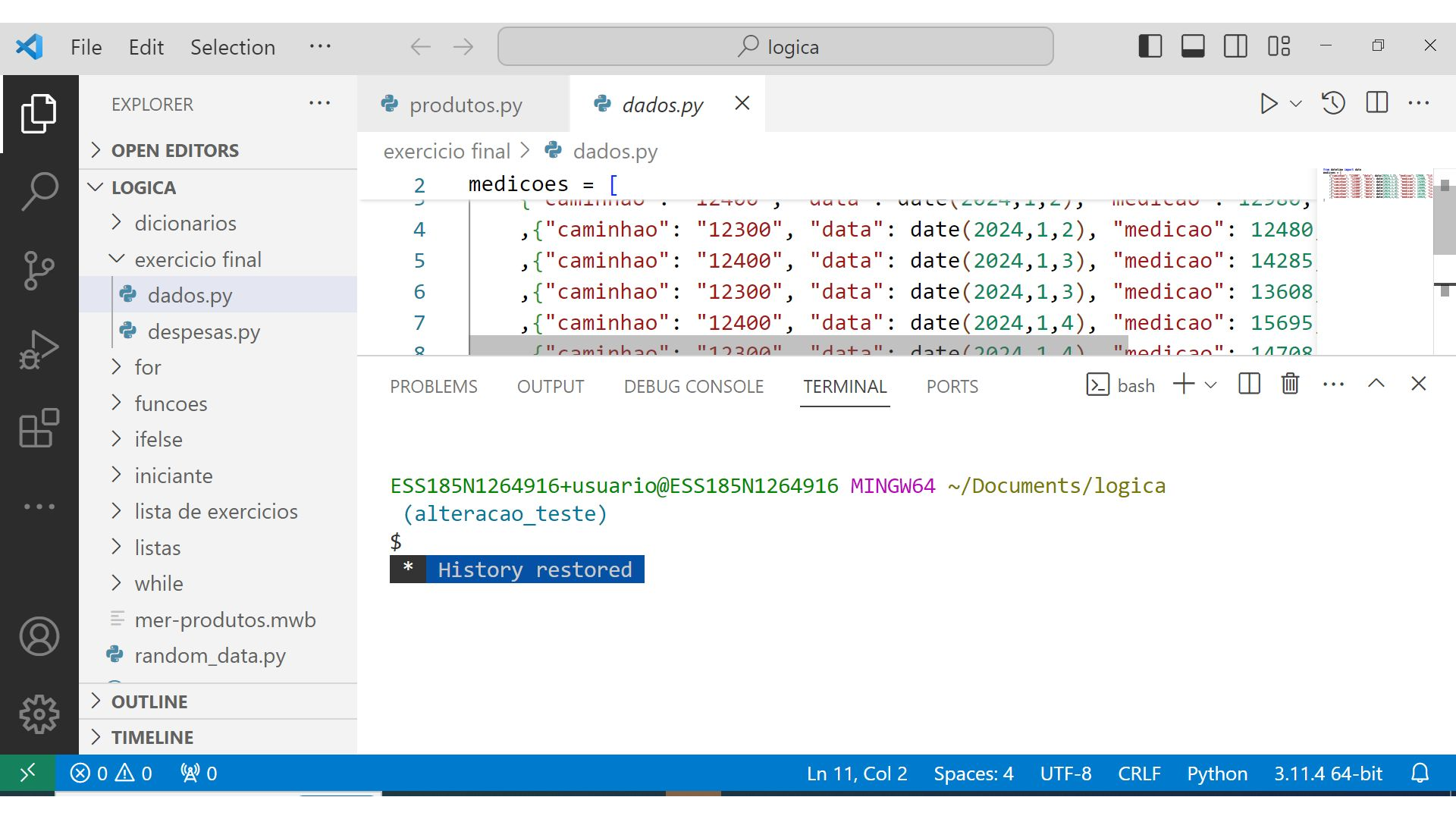
Vamos usar o VS code para escrever nossos códigos.

Não se esqueça de marcar todas as opções no checkbox do instalador

https://code.visualstudio.com/

Por que usar uma IDE?

- Integrated Development Environment.
- Uma IDE é uma ferramenta que integra o ambiente de desenvolvimento.
- Por ela é possível:
  - Gerenciar os arquivos e diretórios.
  - Compilar e testar o projeto.
  - Abrir linha de comando Bash ou Power Shell.
  - Intellisense, autocomplete, separador de cores.


File
Edit
Selection
logica
EXPLORER
OPEN EDITORS
LOGICA
dicionarios
exercicio final
dados.py
despesas.py
for
funcoes
ifelse
iniciante
lista de exercicios
listas
while
mer-produtos.mwb
random_data.py
OUTLINE
TIMELINE
produtos.py
dados.py
exercicio final > dados.py
2 medicoes = [
4 ,{"caminhao": "12300", "data": date(2024,1,2), "medicao": 12480
5 ,{"caminhao": "12400", "data": date(2024,1,3), "medicao": 14285
6 ,{"caminhao": "12300", "data": date(2024,1,3), "medicao": 13608
7 ,{"caminhao": "12400", "data": date(2024,1,4), "medicao": 15695
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash
ESS185N1264916+usuario@ESS185N1264916 MINGW64 ~/Documents/logica
(alteracao_teste)
$
* History restored
0 0 0
Ln 11, Col 2
Spaces: 4
UTF-8
CRLF
Python
3.11.4 64-bit

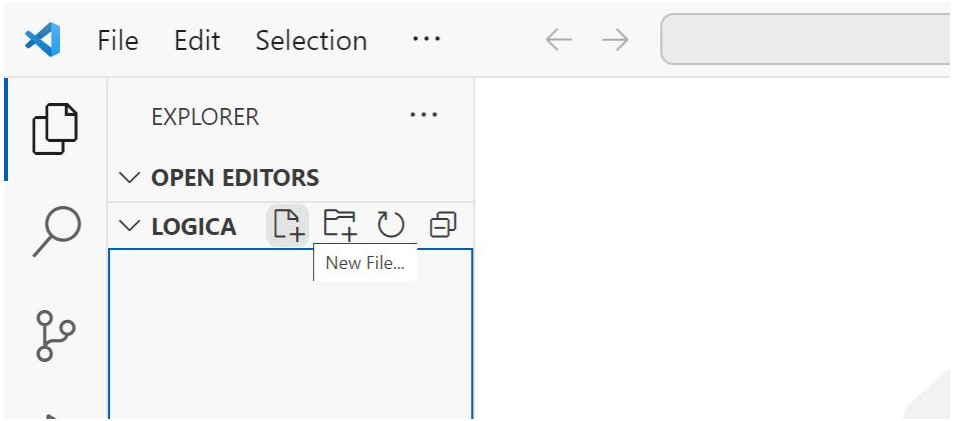
Criando projetos

1. Em uma pasta do Windows crie uma pasta chamada logica.
2. Entre dentro dela
3. Clique com o botão direito do mouse e clique na opção Abrir com o Code.
4.
5. Ou use sua IDE favorita.

Criando um arquivo

Clique no botão New File no painel a esquerda e chame-o de distancia.py.

Não esqueça de usar o .py



Programa distância

distancia.py > ...

```python
#programa distancia
nome = input("Seu nome: ")
origem = input("Cidade de origem: ")
destino = input("Cidade de destino: ")
distancia = float(input("Qual distância vai percorrer? "))
velocidade = float(input("A que velocidade pretente ir? "))
tempo = distancia/velocidade

print(f"{nome}, o tempo estimado entre {origem} e {destino} é de {tempo} horas.")
```

Área do terreno

area.py > ...

```
1    #programa area
2    nome = input("Seu nome: ")
3
4    comprimento = float(input("Qual o comprimento? "))
5    largura = float(input("Qual a largura? "))
6    area = comprimento * largura
7
8    print(f"{nome}, à área é {area}m.")
```

Estruturas de Decisão ou Desvios Condicionais

Se, senao, senao se, escolha….

- Toda linguagem possui suas próprias estruturas.
- A melhor forma de aprendê-las é usando pseudocódigo.

```
se condicao:
    # processamento se verdadeiro
senao:
    # processamento se falso
```

O se testa uma condição, caso ela seja verdadeira ela entra no processamento.

Se não for verdadeira o senao atua para resolver o restante

Blocos de código

Antes de entrar nos detalhes é importante falar sobre blocos de código.

Algumas linguagens (derivadas da linguagem C) costuma criar blocos de código com {}.

```
se (condicao) {
    // processamento
}
```

Blocos de código

O python costuma criar blocos com ‘**:**’ (dois pontos).

Repare que em todas as linguagens tudo que está **indentado** (espaçado com tab) está dentro do bloco.

Clique na palavra destaque para ter uma explicação detalhada.

```
se condicao:
    # código que o bloco executa
```

Operadores Relacionais (comparação)

==	igual	a == 10
!=	diferente	a != 10
>	Maior que	a > 10
<	Menor que	b < 10
>=	Maior ou igual	a >= b
<=	Menor ou igual	b <= b

Operadores Lógicos

and	**e** lógico	a == 10 **and** b != 9	se a for igual à 10 e b diferente de 9, retorna verdadeiro
or	**ou** lógico	a != 10 **or** b == 9	Retorna verdadeiro se qualquer uma das comparações for verdadeira
not	**não** lógico	**not** a	a não tem algum valor válido

Média

Os alunos de uma escola para concluir o semestre precisam entregar um trabalho e fazer duas provas.

A nota de trabalho equivale a 30% da nota final e as outras provas os 70%.

Faça um programa que receba o nome do aluno digitado pelo usuário e as respectivas notas e exiba na tela sua média final e a situação.

media > 50: “aprovado”

media > 35: “recuperação

media <=35: “reprovado”

Use o f”mensagem {variavel}”, para formatar a mensagem

Explicação Detalhada

ENTRADAS:

NOME

PROVA1

PROVA2

TRABALHO

PROCESSAMENTO:

se media >= 50

print(mensagem)

se media > 35

print(mensagem)

senao

print(mensagem)

media = prova1 * 0.35 + prova2 * 0.35 + trabalho * 0.30

SAÍDA:

“Olá, {nome}!

Sua média final foi {media}.

Situação: {situacao}”

Alíquota do imposto de renda

Faça um programa que receba o nome e o salário de um trabalhador e retorne quanto ele pagará de imposto de renda.

A tabela de imposto de renda é:

Até R$ 2112.00 - isento

De 2.112,01 até 2.826,65 - 7.5%

De 2.826,66 até 3.751,05 - 15%

De 3.751,06 até 4.664,68 - 22.5%

Use o elif combinado com o and para fazer os calculos.

if salario <= 2111:
elif salario >=2111 and salario <= 280..

Explicação Detalhada

ENTRADAS:

salario

nome

PROCESSAMENTO:

se salario <= 2111

aliquota = salario * x%

senaose salario > 2111 and salario <= 2800

aliquota = salario * x%

senao

aliquota = salario * x%

SAÍDA:

“Olá, {nome}!

Você pagará uma alíquota de {aliquota}, para seu salario de {salario}”

Validando dados

Se usuário digitar uma nota de 0-100 no nosso programa, com certeza ele será aprovado, precisamos fazer uma validação em nosso algoritmo para impedir que o usuário digite número de 0-10.

Então se prova1 < 0 and prova1 > 10, pedir a nota pro usuário novamente.

Resolução

media4.py > prova1

```
1    #programa média
2    nome = input("Diga seu nome: ")
3    prova1 = float(input("Nota da prova 1: "))
4    if prova1 < 0 or prova1 > 10: # Valida prova1
5        prova1 = float(input("Digite o valor de 0-10 pra prova1: "))
6
```

Pegando dados de contato

Faça um programa que receba o nome completo, assunto, email e uma mensagem de texto de um usuário.

E exiba ao final a seguinte mensagem:

Nova mensagem! - {assunto}

Olá,

Gostaria de falar sobre: {mensagem}

Atenciosamente!

{nome}

{email}.

Listas

Listas

- São estruturas de dados que agrupam valores em sequência;
- São acessadas por índices;
- E podem ser modificadas, adicionando ou removendo elementos.
- Os elementos das listas podem ser heterogêneos com

```
alunos = ["Alberto", "Neto", "Sandra", "Paulo", "Priscila"]
```

- Listas são representadas com colchetes []

Acessando elementos de uma lista

- Podemos acessar um elemento através do seu índice;
- Todo índice começa na posição 0, logo o primeiro elemento está no índice 0.
- Quer dizer que para acessar um elemento de uma lista basta escrever o seu nome seguido de ‘[0]‘ (índice dentro do colchete).

lista[0]

Exemplo

stas > lista_basico.py > ...

```
1  alunos = ["João", "Maria", "Pedro", "Paulo", "Tiago"]
2  print(alunos) # exibe todos os alunos
3  print(alunos[2]) # exibe somente o Pedro
4  print(alunos[-1]) # exibe o Tiago
```

Também podemos exibir somente o que queremos

Para isso é possível passar o intervalo de índices como em lista[0:9].

```
1   alunos = ["João", "Maria", "Pedro", "Paulo", "Tiago"]
2   print(alunos) # exibe todos os alunos
3   print(alunos[2]) # exibe somente o Pedro
4   print(alunos[-1]) # exibe o Tiago
5   print(alunos[1:3]) # exibe Maria e Pedro
```

Adicionando e removendo elementos da lista

- Podemos adicionar um elemento novo com o método **.append(elemento).**
- E remover com a função del(lista[1])

```
7    alunos.append("Patrícia") # Adiciona a patrícia
8    del(alunos[1]) # Remove a Maria
9    alunos.remove("Paulo") # Remove a primeira palavra
```

Exercícios

Faça um programa para receber do usuário o nome e a média de 5 alunos.

O programa deve armazenar esses dados em uma lista de nomes e outra de notas.

Colocando as coisas em ordem…

- As vezes é necessário ordenar uma lista de dados.
- Podemos usar a função sorted ou o método .sorted().

```
11  # ordenar uma lista
12  print(sorted(alunos))
13  alunos.sort() # ou...
14  print(alunos)
```

Exercício - Ordene as listas

Faça um programa para pegar o nome de 10 animais diferentes de forma aleatória.

Depois exiba os animais em ordem alfabética.

Saber o índice de uma sentença

- O método index retorna o indice do primeiro elemento chamado Pedro.

```
16    # indice da sentença
17    print(alunos.index("Pedro"))
```

Inserir elemento em uma determinada posição

Podemos usar o método insert para adicionar um elemento em uma posição específica.

```
19    # Inserindo em uma posição
20    alunos.insert(3, "João")
21    print(alunos)
```

Contar elementos de uma lista

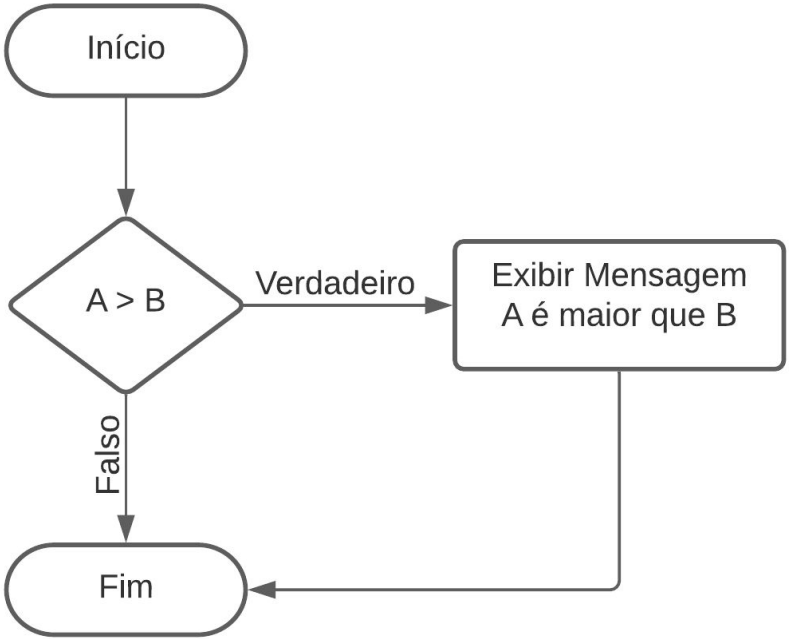
Podemos usar a função len(lista) do python para contar as listas

```
23    #Contando elementos
24    print("Total de elementos", len(alunos))
```

Laços de Repetição

Laços

- São responsáveis por fazer iterações ou repetir um trecho de código;
- Podem ser infinitos;
- Podem ser iterativos, ou seja, até um determinado número de vezes;



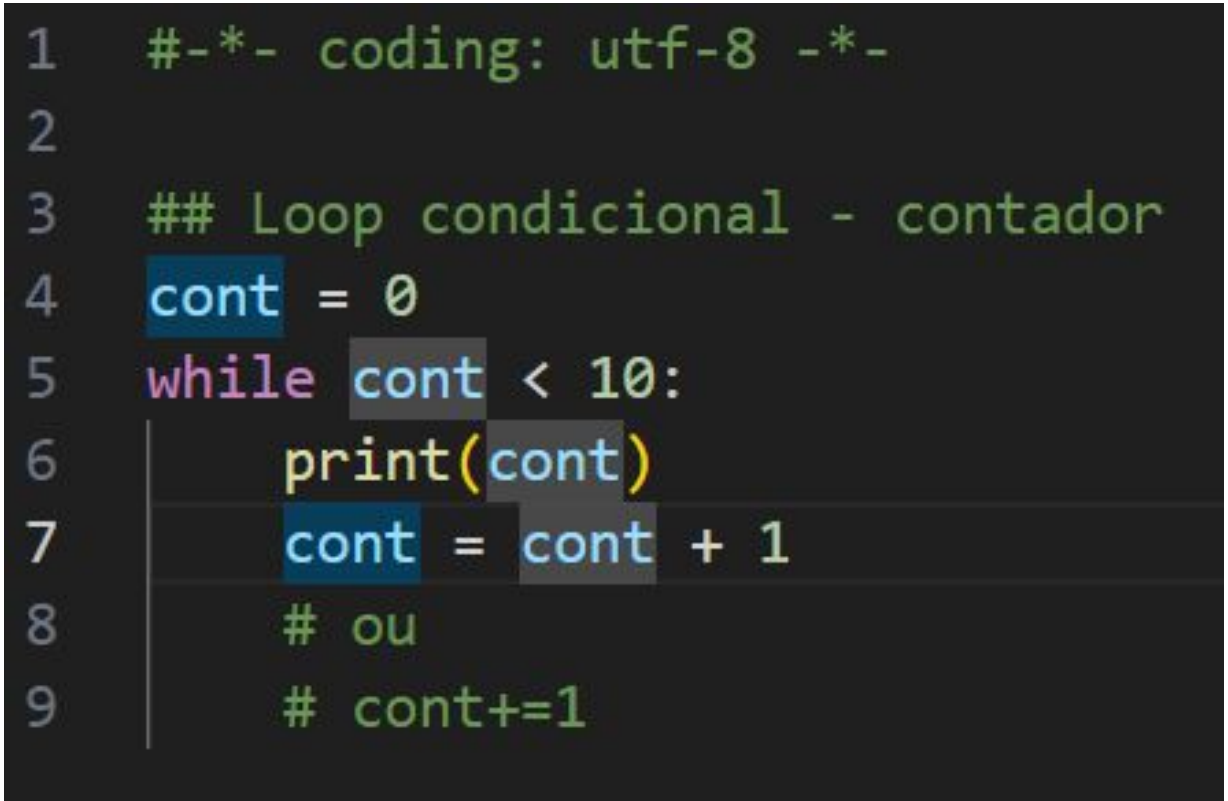
While (enquanto)

- Usado quando não sabemos exatamente até quando um trecho de código deve ser executado;
- Como em:
  - Menus;
  - Leitura de arquivos;
  - Condições onde o usuário dirá quando deve parar;
  - Programas que devem rodar até que ocorra algum erro;

Loop infinito

```python
#-*- coding: utf-8 -*-

## Loop infinito
while True:
    print("="*40)
    print(""" MENU
          1 - Notas;
          2 - Faltas;
          0 - Sair;""")
    menu = int(input(": "))
    if menu == 1:
        print("Notas....")
    elif menu == 2:
        print("Faltas...")
    elif menu == 0:
        print("Saindo...")
        break
    else:
        print("Você precisa digitar opção válida!\n\n")
```

Loop condicional



Usando loop nas validações da média

Um bom exemplo de loop é validar os dados que o usuário digita.

O código possui alguns problemas, pois o usuário pode

media4.py > prova1

```
1   #programa média
2   nome = input("Diga seu nome: ")
3   prova1 = float(input("Nota da prova 1: "))
4   if prova1 < 0 or prova1 > 10: # Valida prova1
5       prova1 = float(input("Digite o valor de 0-10 pra prova1: "))
6
```

Usando loop nas validações da média

Uma solução é usar o loop while condicional para cada valor

```
3
4   prova1 = float(input("Nota da prova 1: "))
5   while prova1 < 0 or prova1 > 10: # Valida prova1
6       prova1 = float(input("Digite o valor de 0-10 pra prova1: "))
7
```

Repare que apenas trocamos if por while

```
Diga seu nome: Rodrigo
Nota da prova 1: 11
Digite o valor de 0-10 pra prova1: -1
Digite o valor de 0-10 pra prova1: 10.1
Digite o valor de 0-10 pra prova1: 4
Nota da prova 2:
```

Alterando os programas anteriores

1) Faça um programa que receba o nome e a nota (0-10) de 3 alunos diferentes e exiba na tela a seguinte mensagem.

   “O aluno {nome} teve a maior média {media}”

2) Faça um programa que receba números inteiros e pare somente quando o número atual for menor que o anterior.

Tabuada

Faça um programa que receba um número inteiro e exiba a sua tabuada do 1 ao 10 como abaixo.

2 x 1 = 2

2 x 3 = 6

…

2 x 10 = 20

Quanto tempo leva

Supondo que a população de um país A seja da ordem de 80000 habitantes com uma taxa anual de crescimento de 3% e que a população de B seja 200000 habitantes com uma taxa de crescimento de 1.5%. Faça um programa que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas as taxas de crescimento.

Iterando listas com while

```
alunos = ["Alberto", "Neto", "Sandra", "Paulo", "Priscila"]
cont = len(alunos) # conta elementos
while cont > 0:
    idx = cont-len(alunos)
    print(alunos[idx])
    cont-=1
```

Loop for

- É usado quando sabemos quando a iteração irá terminar;
- Geralmente usamos para iterar listas de dados ou fazer contadores;

For in range

```
1   #-*- coding: utf-8 -*-
2
3   ## Loop condicional - contador
4   for p in range(10):
5       print(p)
```

Exercícios

1. Faça um programa que exiba uma sequência de números ímpares de 0 a 100;
2. Faça um programa para exibir o ano atual e sua idade até 2040.
    a. Exemplo 2024 - 23 anos, 2025 - 24 anos….
3. Faça um programa para exibir a tabuada do 10.
    a. Exemplo 10x1 = 10, 10x2=20…

Validando a senha

1. Faça um programa que armazene uma senha em uma variável e de ao usuário 3 tentativas para acertar a senha. O programa deve exibir o número de tentativas que restam para ele acertar até chegar a 3. Se ele acertar dentro das 3 tentativas exibir uma mensagem “Logado com sucesso!”

Exemplo de saída

```
Qual é a senha: 0
Senha incorreta!
Você tem mais 2
Qual é a senha: 33
Senha incorreta!
Você tem mais 1
Qual é a senha: 9
Senha incorreta!
Tentativas esgotadas.
```

```
Qual é a senha: abacate
Senha incorreta!
Você tem mais 2
Qual é a senha: abacaxi
Logado com sucesso!
```

For iterator

- Todo for do python é um iterator, ou seja ele sempre vai percorrer uma lista.
- Mesmo o for p in range(10), range(10) retorna [0, 1, …, 9], uma lista.

```
1  #-*- coding: utf-8 -*-
2
3  ## Loop iterator
4  for p in ["Alberto", "Neto", "Sandra", "Paulo", "Priscila"]:
5      print(p)
```

Iterando listas com for

```
## Loop iterator
alunos = ["Alberto", "Neto", "Sandra", "Paulo", "Priscila"]
for p in alunos:
    print(p)
```

Exercícios Percorrendo Listas

- Faça um programa que exiba o elemento da lista antecedido pelo seu índice, você pode usar a função range combinada com a função len.

Exemplo:

0 - João

1 - Paulo

2 - Riana

Exercícios Percorrendo Listas

Faça um programa que armazene em uma lista as compras e em outra os valores de cada item comprado para exibi los da seguinte maneira.

```
Martelo - R$ 29.9
Chave Fenda - R$ 10.99
Serrote - R$ 41.49
Macete - R$ 19
```

Exercícios - Apagando dados

Dado uma lista aleatória:

carros = [‘Vectra’, ‘Astra’, ‘Palio’, ‘Saveiro’, ‘Maserati’]

Elabore um programa que exiba todos os elementos da lista com seu índice logo início. Depois receba um número do usuário para remover um item da lista e exiba novamente todos os elementos da lista.

Laços na prática

- Em se tratando de sistemas de informação com Python o loop mais usado será o loop for;
- Isso porque geralmente já se sabe a quantidade de dados a processar.

CRIE UM ARQUIVO .PY COM NOME **lista_alunos.py**

Ex os.py

```python
#-*- coding: utf-8 -*-

alunos = []
while True:
    print("""Menu
        1 - Adicionar aluno
        2 - Remover aluno
        3 - Exibir aluno
        0 - Sair
        """)
    menu = int(input(": "))
    if menu == 1:
        print("Adicionar aluno!")
        nome = input("Nome do aluno: ")
        alunos.append(nome)
```

Parte2

```
16      elif menu == 2:
17          for idx, a in enumerate(alunos):
18              print(idx, a)
19          idx = int(input("Para remover selecione um index: "))
20          del alunos[idx]
21      elif menu == 3:
22          print("Lista de alunos")
23          for idx, a in enumerate(alunos):
24              print(idx, a)
25      elif menu == 0:
26          print("Saindo...")
27          break
28      else:
29          print("Você deve digitar uma opção do menu\n")
```

Quanto tempo leva

Um caracol sobe em uma cisterna cerca de 1m/h mas a cada 2 horas ele regride 50cm. Sabendo que o caracol já subiu 10m dos 22 metros de profundidade da cisterna.

Sabendo elabore um programa que exiba quanto tempo (hora por hora) o caracol leva para chegar ao topo.

Use um loop while ou for para resolver esse problema.

Exercício ordem

Faça um programa que receba do usuário o nome, data de nascimento e altura de 10 alunos.

Ao final permita que o usuário exiba esses nomes ordenados por ordem: alfabética, nascimento ou altura.

Você pode usar uma lista auxiliar combinada com if para ordem alfabética ou usar os métodos de classe conforme documentação do python.

Tuplas

- São listas imutáveis, ou seja, não pode remover ou adicionar elementos.

```
dias = ('Segunda-feira', 'Terça-feira'
        , 'Quarta-feira', 'Quinta-feira'
        , 'Sexta-feira', 'Sábado'
        , 'Domingo')
```

Dicionários

- Existem estruturas de dados que são compostas.
- Ou seja, elas têm chaves e valores para defini-las;
- Diferente das listas os dicionários são acessados por suas chaves, que podem ser de qualquer tipo de dado do python.
- Dicionários são representados entre chaves ‘{}’

```
alunos = {
    1: "Alberto"
    ,2: "Neto"
    ,3: "Sandra"
    ,4: "Paulo"
    ,5: "Priscila"
}
```

Manipulando dicionários

- Diferente das listas onde é preciso saber qual o índice sequencial.
- O dicionário é manipulado pelas chaves.
- As chaves podem ser qualquer valor, qualquer tipo de dado ou objeto.

```
alunos = {
    1: "Alberto"
    ,2: "Neto"
    ,3: "Sandra"
    ,4: "Paulo"
    ,5: "Priscila"
}
print(alunos[1])
alunos["novo"] = "Silas"
print(alunos)
del alunos["novo"]
```

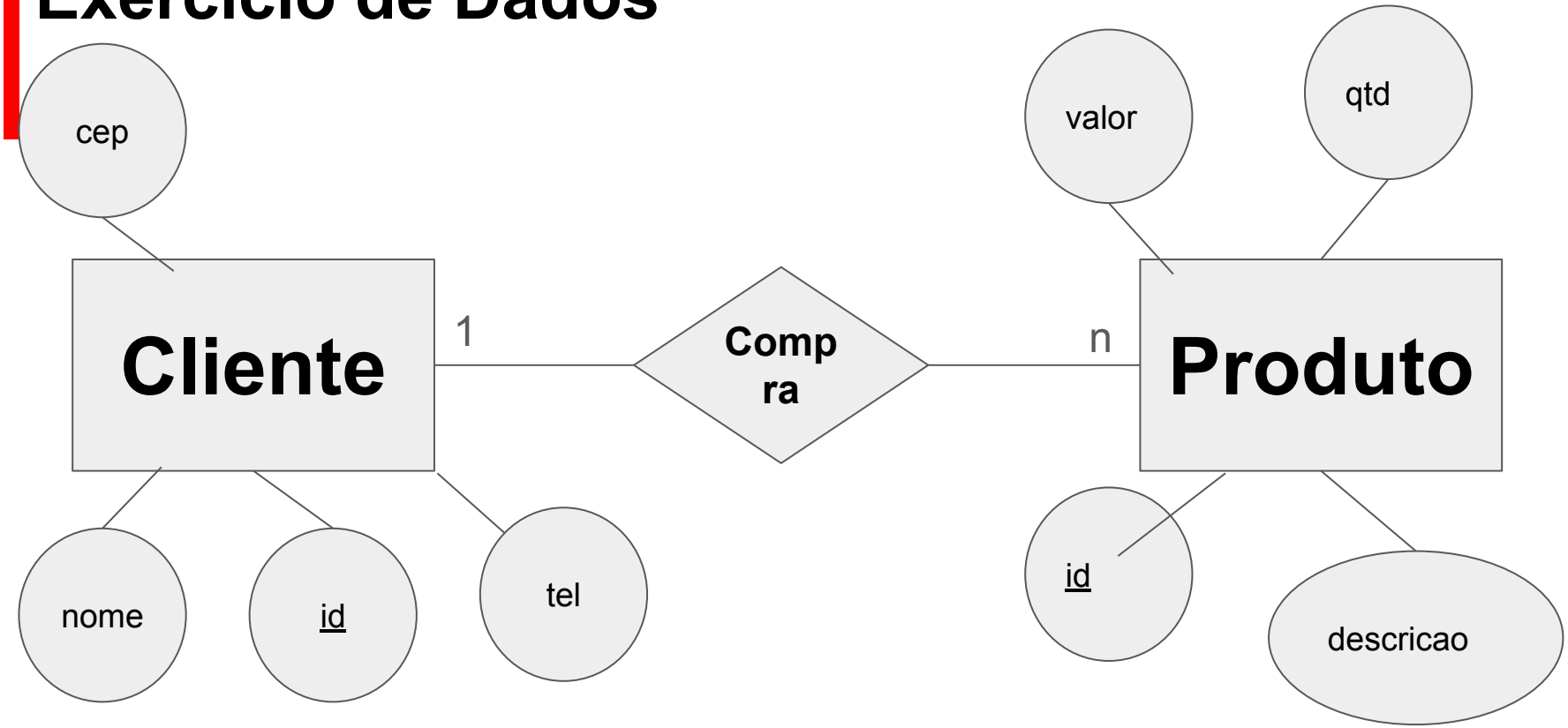
Iterando dicionários

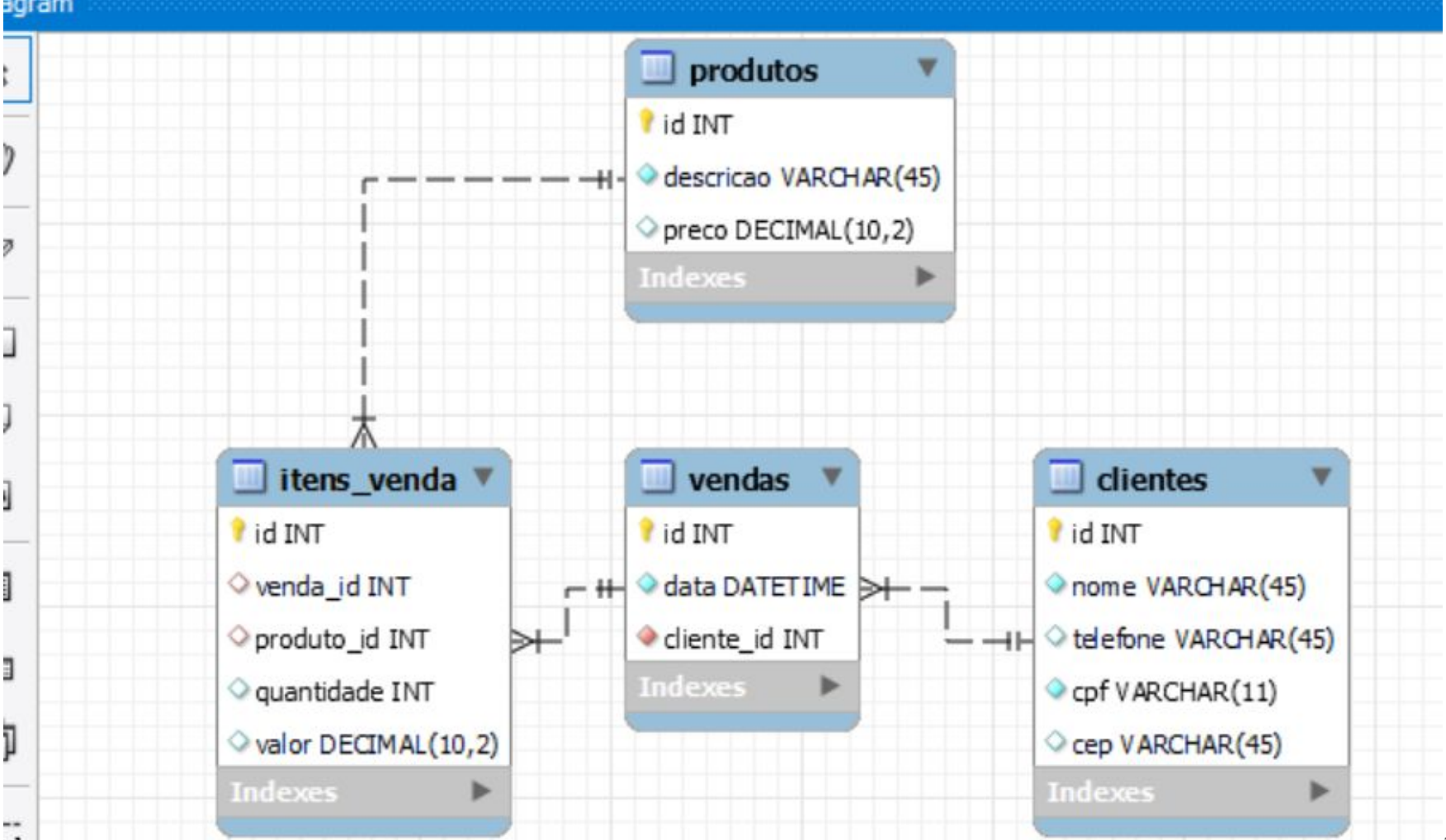
```python
for chave, valor in alunos.items():
    print(chave, valor)
# ou
for item in alunos.items():
    print(item[0], item[1])
```

Iterando dicionários

```
#ou
for p in alunos.keys():
    print(alunos[p])
#ou
for p in alunos.values():
    print(p)
```

Exercício de Dados




produtos
id INT
descricao VARCHAR(45)
preco DECIMAL(10,2)
Indexes
itens_venda
id INT
venda_id INT
produto_id INT
quantidade INT
valor DECIMAL(10,2)
Indexes
vendas
id INT
data DATETIME
cliente_id INT
Indexes
clientes
id INT
nome VARCHAR(45)
telefone VARCHAR(45)
cpf VARCHAR(11)
cep VARCHAR(45)
Indexes

Exercício de dados

Usando os conhecimentos sobre dicionário que possui, elabore um programa que exiba na tela uma lista de produtos pré-cadastrados em uma lista de dicionários e depois permita que o usuário compre um dos produtos.

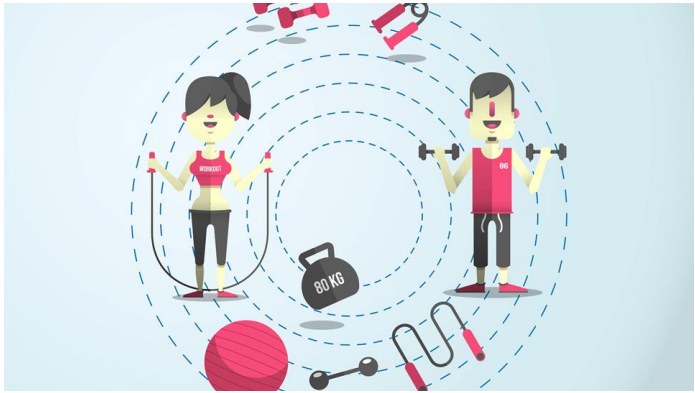
Para comprar o produto ele deve se cadastrar no sistema.

Use os exemplos das aulas anteriores para resolver o problema.

NOTA: você precisará de um terceiro dicionário para relacionar produto e cliente

$$f(x) = ax + b$$

Funções



Funções

- São trechos de código prontos que podem ser “invocados” a qualquer momento no código;
- Cada linguagem possui algumas bibliotecas de funções próprias;
- Além de permitir criar suas próprias funções.

Funções built-in do Python

- São funções próprias da linguagem python que não precisam de um import:
  - Intervalos de números;
  - Enumeradores;
  - Datas e Horas;
  - Leitura e gravação de arquivos;
  - Strings e etc.
- **TODA FUNÇÃO É CARACTERIZADA** por seu nome seguido de parênteses;
- Entre parênteses podem vir parâmetros separados por vírgula.
- nome() ou nome(a, b)

https://docs.python.org/pt-br/3.6/library/functions.html

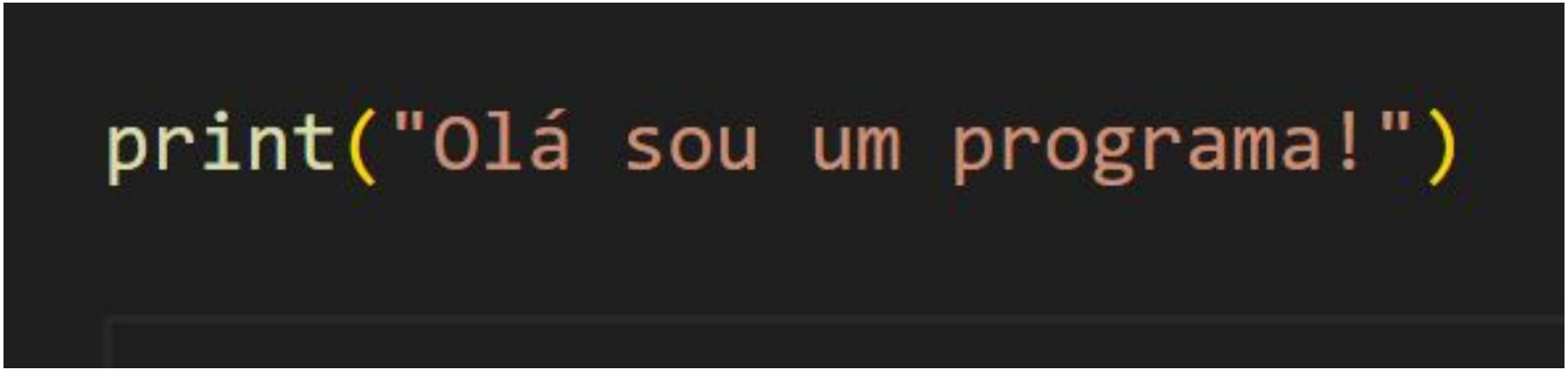
		Funções Built-in		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Algumas funções úteis para o cotidiano

- range
- list
- sorted
- dict
- int
- min
- float
- sum
- max
- type
- len
- eval

print

- Serve para exibir informações no console (terminal) ou linha de comando



range

- Função que cria um intervalo de números;
- Combinamos geralmente com loop for ou podemos converter em uma lista de números;

Forma de usar

```
#range (gera uma lista)
print(range(10))
```

✓ 0.0s

```
range(0, 10)
```

```
#Combine com list
print(list(range(10)))
```

✓ 0.0s

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

sorted

- Server para ordenar uma lista.

```
3   alunos = ["Batman", "Homem Aranha", "Coringa", "Cegonha"]
4   print(sorted(alunos))
5   print(sorted(alunos, reverse=True))
```

Forma de usar

```
#Ordene a lista
numeros = [0, 8, 2, 7, 9, 6, 5, 3, 1, 4]
print(sorted(numeros))
```

✓ 0.0s

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Forma de usar

```
#Converta números para string
print(str(10.9), str(10))
```

✓ 0.0s

```
10.9 10
```

Forma de usar

```
#Converta strings para numeros inteiros ou flutuante para inteiro
print(int("10"), int(10.2))
```

✓ 0.0s

```
10 10
```

len

- Conta elementos de uma lista, dicionário, tupla

```
3   notas = [10, 9, 8]
4   # Use
5   print(len(notas))
6   print(len(range(5)))
7   print(len(notas)>4)
```

Forma de usar

```
#conte listas, dicionários e caracteres
numeros = [0, 8, 2, 7, 9, 6, 5, 3, 1, 4]
print(len("10"), len("seu nome"), len(numeros))
```

✓ 0.0s

```
2 8 10
```

sum

- Serve para fazer somatórios

```
notas = [10, 9, 8]
# Use
print(sum(notas))
# Combine para média
print(sum(notas)/len(notas))
```

Forma de usar

```
#Somando valores
numeros = [0, 8, 2, 7, 9, 6, 5, 3, 1, 4]
print(sum(numeros))
```

✓ 0.0s

```
45
```

Forma de usar

```
#Valor máximo e minimo
numeros = [0, 8, 2, 7, 9, 6, 5, 3, 1, 4]
print(max(numeros), min(numeros))
```

✓ 0.0s

```
9 0
```

type

- Use para descobrir o tipo de dado

```
5   print(type(notas))
6   # >>> <class 'list'>
7   print(type(range(5)))
8   # >>> <class 'range'>
9   print(type(len(notas)>4))
0   # >>> <class 'bool'>
```

Forma de usar

```
#Descobrindo o tipo da variável
print(type(10), type([9, 8]), type(sum([9, 10])), type('10')==type(10))
```

✓ 0.0s

```
<class 'int'> <class 'list'> <class 'int'> False
```

Funções que na verdade são tipos de dados

- int
- float
- list
- dict
- str
- bool

Enumerate

```
alunos = ["Batman", "Robin", "ScoobyDoo"]
for idx,p in enumerate(alunos):
    print(idx,p)
```

Funções importadas

- As vezes é preciso importar funções da biblioteca do python pois a built-in não atende nossa necessidade.
- Para isso precisamos usar no topo do nosso arquivo o comando **import** seguido do nome da biblioteca.
- Uma biblioteca é um conjunto de código pronto que podemos utilizar no código principal.

Trabalhando com datas

```
# Leia-se do arquivo datetime
# importe a classe date
from datetime import date

# Data simples
hoje = "25/07/2023"
print(type(hoje), hoje)
hoje = date.today()
print(type(hoje), hoje)
```

Mais sobre datas clique aqui

Trabalhando com datas

```
hoje = date.today()
print(type(hoje), hoje)
print(f"Hoje é dia {hoje.day} \
      de {hoje.month} de {hoje.year}")
```

Trabalhando com datas - Semanas

```
dias = ('Segunda-feira', 'Terça-feira'
        , 'Quarta-feira', 'Quinta-feira'
        , 'Sexta-feira', 'Sábado'
        , 'Domingo')
print(f"{hoje.weekday()} {hoje}")
```

Definindo nossas funções

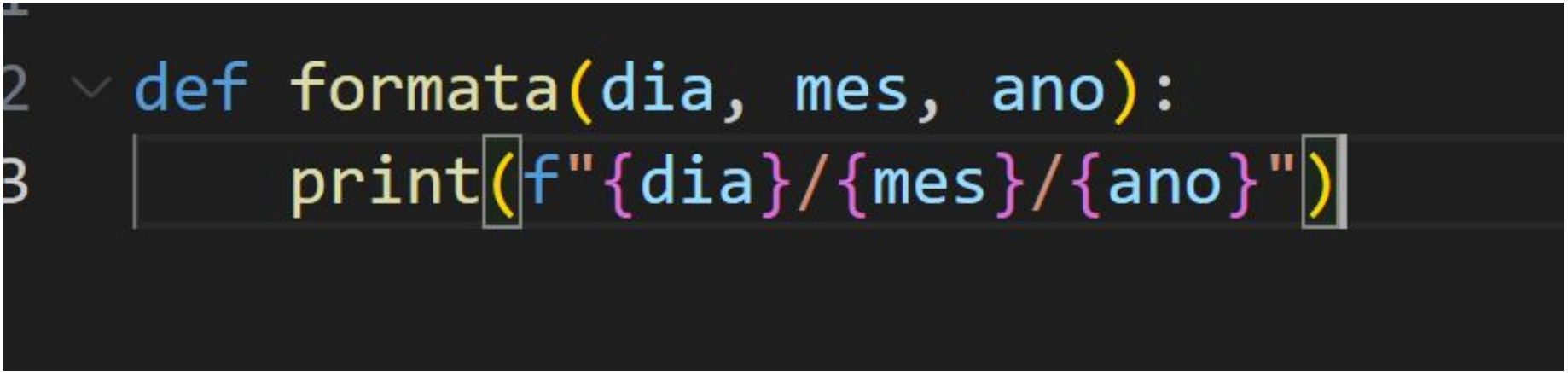
- O Python como toda linguagem nos permite criar nossas próprias funções

```
def media(dados=[]):
    """ dados: Recebe lista a somar
        return: somatorio/elementos
    """
    return sum(dados)/len(dados)
valores = [10, 9, 7]
print(f"A média entre 10, 9, 7 é {media(valores)}")
```

Funções podem não receber parâmetro nenhum

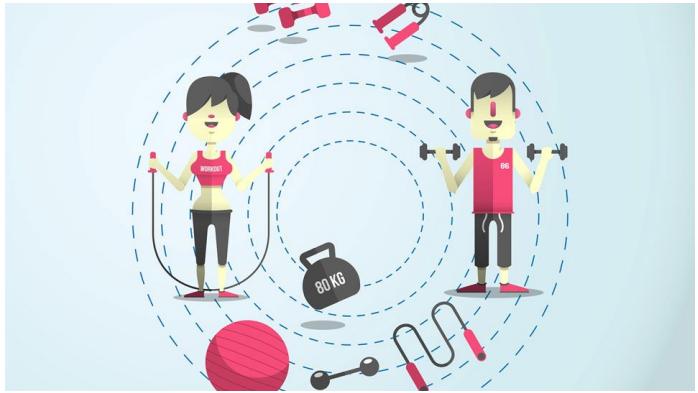
```
def menu():
    return """
        Menu
        1 - Opção
        2 - Opção
        3 - Sair
    """

print(menu())
```

Funções podem não ter retorno



$$f(x) = ax + b$$

Criando minhas próprias funções



Por que criar nossas funções

Toda linguagem de programação permite que o desenvolvedor crie trechos de código prontos para serem chamados a qualquer hora no código.

Além das funções prontas como visto anteriormente é fundamental saber como criar uma função.

O básico de uma função

```
funcao nome_da_funcao() {
    // processamento
    //retorno
}
```

Toda função tem um nome e código a ser processado que pode ou não ser retornado.

Função no python

O python define suas funções com o def seguido do nome e parênteses.

Repare que ele trata uma função como qualquer outro bloco de código, por tanto é necessário indentar (tab) o código que será processado pela função

```
def mensagem():
    print("Só exibe essa mensagem")
```

‘Chamando uma função’

A função acima pode ser chamada pelo seu nome seguido de parênteses como no exemplo.

```
def mensagem():
    print("Só exibe essa mensagem")

mensagem()
```

Funções com retorno

É uma boa prática deixar que a função print exiba os valores e as funções apenas retornem os valores processados.

Repare que agora passamos a mensagem() dentro do print()

```
def mensage():
    return "Só exibe essa mensagem"

print(mensagem())
```

Funções com retorno

Função para exibir um menu

```
1  def menu():
2      return """
3          Menu
4          1 - Opção
5          2 - Opção
6          3 - Sair
7      """
8  print(menu())
```

Funções que processam algo

As funções são trechos de código que podem ser reaproveitados por exemplo a função de mensagem podemos usá-la para formatar campos de moeda.

Funções que processam algo

```
def formato_moeda():
    valor = 1390.99
    valor = "R$ {:.2f}".format(valor)
    # troca ponto por virgula
    valor = valor.replace(".", ",")
    return f"{valor}"
print(formato_moeda())
```

As funções também pode receber parâmetros

```
def formato_moeda(valor):
    valor = "R$ {:.2f}".format(valor)
    # troca ponto por virgula
    valor = valor.replace(".", ",")
    return f"{valor}"
print(formato_moeda(1390.99))
```

As funções também pode receber parâmetros

```
1  def media(dados=[]):
2      """ dados: Recebe lista a somar
3          return: somatorio/elementos
4      """
5      return sum(dados)/len(dados)
6
7  valores = [10, 9, 7]
8  print(f"A média entre 10, 9, 7 é {media(valores)}")
```

As funções também pode receber mais de um parâmetro

Cálculo de área

```
def area_total(largura, altura):
    return largura*altura


area = area_total(10, 9)
print("À área total é {area}m²")
```

As funções também pode receber mais de um parâmetro

Formatando datas de inteiro

```
1  def formata(dia, mes, ano):
2      return f"{dia}/{mes}/{ano}"
3  print(formata(10, 1, 2024))
```

Onde eu crio minhas funções?

É fundamental que as funções sejam declaradas antes do código que precisam delas, como visto nos exemplos acima.

Mas também é possível criar uma biblioteca para armazenar e poder importar as funções que precisamos mais tarde.

Bibliotecas

Os arquivos que contém as funções a serem chamadas quando precisamos são chamadas de bibliotecas, geralmente criamos um diretório chamado lib e colocamos nossos arquivos .py lá dentro.

Aqui vamos apenas criar um arquivo chamado funcoes.py e colocar as funções: **`media(dados=[]), formato_moeda(valor), menu(), area_total(largura, altura) e formata(dia, mes, ano).`**

Bibliotecas importando essas funções

Há várias formas de importar uma biblioteca no python.

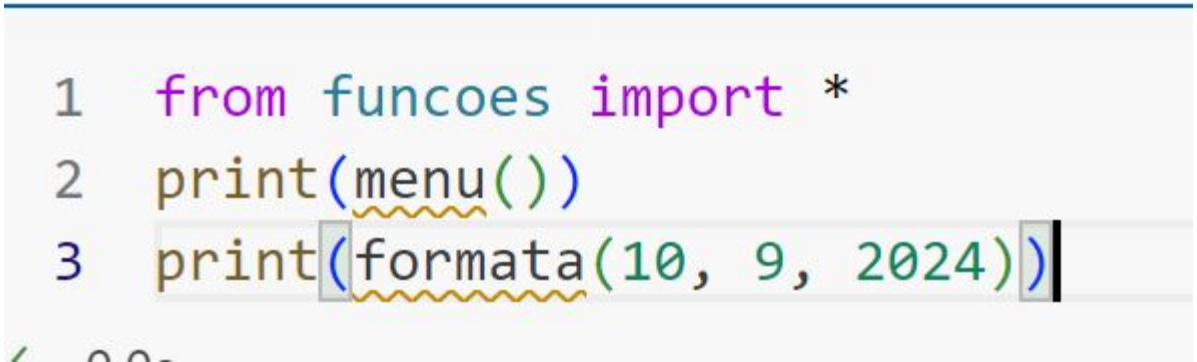
A mais básica é usar o import nome_biblioteca e usar o nome.funcao()

```
1  # import tudo (*)
2  import funcoes
3
4  print(funcoes.menu())
5  print(funcoes.formata(10, 9, 2024))
```

Bibliotecas importando essas funções

OU….

Usando o from com *, repare que não precisamos mais usar o nome do import



from import vs só import

O import é a importação genérica e completa há casos onde precisamos usá-lo, normalmente para identificar de onde vem a função e também quando precisamos de todas as funções do arquivo.

O from import é usado quando queremos apenas uma função específica do arquivo, logo from import arquivo * não é aconcelhado

Como usar o from import então?

Como dito form import é usado para apenas uma função específica como em.

Repare que se tentar usar qualquer outra função do arquivo não será possível

```
1  from funcoes import menu
2  print(menu())
```

0.0s

Como usar o from import então?

Para resolver isso vamos passar cada função separada por vírgula no import

```
1  from funcoes import menu, formata
2  print(menu())
3  print(formata(10, 9, 2024))
```

Exercícios

Dado o dicionário random_data.py, e o seu arquivo funcoes.py crie em um terceiro arquivo main.py um programa para exibir para o usuário conforme ele solicitar em um menu:

A quantidade de pessoas maiores de 50 anos.

A quantidade de pessoas que ganham mais de 2000 mil

O nome, salário, idade e profissão das 3 pessoas com maiores salários

A média salarial de cada profissão

Orientação a Objetos

Orientação a Objetos????

- Em Python tudo é objeto.
- Até os tipos primitivos.

```
type(10)
<class 'int'>
type([])
<class 'list'>
type('p')
<class 'str'>
```

Mas o que objeto tem haver com classe?!?!?

- O conceito de orientação a objetos começou com a necessidade de aproximar ainda mais a programação do mundo real.
- O predecessor disso foram as estruturas de dados compostas como dicionários.
- Em resumo objetos do mundo real possuem atributos e métodos.

Atributos??

Objeto da classe Computador:

processador = Intel Core i7

memória = 16 GB

SSD = 512 GB

Objeto da classe Pessoa:

nome = “Fulano”

idade = 18

peso = 58

Atributos??

Objeto da classe Aluno:

turma = 1A

escola = Escola Legal

SSD = 512 GB

Objeto da classe Pessoa:

nome = “Fulano”

idade = 18

peso = 58

Objetos são na verdade classes

- Todo objeto pertence a uma classe (objeto).
- E seus atributos são como se fossem variáveis.
- A direita um exemplo em python

```
class Pessoa:
    nome = “Fulano”
    idade = 18
    peso = 58
```

Boas práticas para definir classes

- Para diferenciar uma classe de uma função ou objeto sempre escrevemos seu nome iniciado em letra maiúscula.
- Palavras compostas devem ser concatenadas com a segunda palavra iniciando em maiúscula, ItensVenda.
- Os nomes dos atributos seguem o padrão das variávies, sempre minúscula.
- Palavras compostas dos atributos devem usar ‘_’ sublinha para separar palavras, atributo_01.

Como usar classes?

- Seguindo o mesmo raciocínio das funções, as classes precisam ser definidas antes do código que as utilizam.
- E como as funções elas são chamadas usando o seu nome seguido de parênteses.
- Com uma diferença, elas são instanciadas dentro de uma variável.

Instância????

- Uma instância é a alocação de memória da classe de objeto.
- Sempre que vamos usar uma classe precisamos alocar memória para ela, chamamos esse processo de instância.

Instância da classe Pessoa

```
class Pessoa:
    nome = “Fulano”
    idade = 18
    peso = 58
```

```
pessoa = Pessoa()
# accessando atributos
print(pessoa.nome)
# >>> Fulano
#mudando valor de atributo
pessoa.idade = 48
```

O que esse ponto faz ali???

- Depois de instanciar a classe podemos acessar seus atributos usando o nome da instância.nome_do_atributo.
- Dessa forma podemos organizar nossos dados de uma forma melhor que um dicionário.

Comparando

```
pessoa = {
    “nome”: “Fulano”,
    “idade”: 18,
    “peso”: 68
}
pessoa[“nome”]
```

```
class Pessoa:
    nome = “Fulano”
    idade = 18
    peso = 58
pessoa = Pessoa()
pessoa.nome
```

Métodos de classe

- Métodos são como funções, porém são específicas da classe.
- Seguindo nosso exemplo uma pessoa pode andar, correr, falar, pular….
- Os nossos objetos também podem fazer o que definimos em um método.

Métodos de classe

- Vamos assumir que você precise definir uma estrutura de dado para cálculo de IMC (índice de massa corpórea).
- Para isso você precisa da altura, do peso e do nome.
- Essa classe terá um método que devolva calculado o IMC.
- Você sabe que imc = peso/altura²
- A definição da classe ficaria assim:

Método

Repare que precisamos passar um parâmetro dentro do método que não aparece quando a classe é instanciada.

O self é a instância interna da classe Pessoa.

```
class Pessoa:
    nome = “Fulano”
    altura = 1.80
    peso = 58
    def imc(self):
        imc = self.peso/self.altura**2
pessoa = Pessoa()
pessoa.imc()
```

Na prática….

```
1   # Classe pessoa
2
3   class Pessoa:
4       nome = "Fulano"
5       altura = 1.8
6       peso = 68
7
8       def imc(self):
9           imc = self.peso/self.altura**2
10          return imc
11
12  pessoa = Pessoa()
13  print(pessoa.imc())
```

Exercício