VALIDAÇÃO DE ENTRADA DE DADOS

- Podemos utilizar funções para criar uma infinidade de coisas, uma das coisas para a qual podemos criar funções é para validar a entrada de dados do usuário, reduzindo assim, possíveis erros no sistema.
- Imagine que você tenha que validar uma entrada de dados específica, em vez de ficar sempre escrevendo o código de validação, você pode criar uma função e chamá-la sempre que necessário.

EXEMPLO

Vamos supor que você precise solicitar que o usuário informe um número inteiro dentro de uma faixa. Você pode criar uma função que receberá os três números e fará esta validação.

O `isdigit()` método retorna True se todos os caracteres forem dígitos, caso contrário, False.

```
def validar_faixa(numero, inicio, fim):
    if numero.isdigit():
        if int(numero) < int(inicio) or int(numero) > int(fim):
            print(f"Valor inválido! Informe um número inteiro entre
{inicio} e {fim}.")
        else:
            return True
    else:
        print(f"Número inválido! Informe um número inteiro entre
{inicio} e {fim}.")
while True:
    resposta = input("Informe um número inteiro entre 1 e 10: ")
    if validar_faixa(resposta, 1, 10):
        break
```

EXEMPLO 2

- Vamos validar um limite máximo de caracteres de um texto informado pelo usuário.

```
def validar_tamanho(texto, maximo):
    if len(texto) > maximo:
        print(f"O texto deve conter no máximo {maximo}
caracteres!")
    else:
        return True
while True:
    texto = input("Informe um texto de no máximo 20
caracteres: ")
    if validar_tamanho(texto, 20):
        break
```

EXEMPLO 3

```
def valida_elementos(texto, lista):
    if texto.upper() in lista:
        return True
    else:
        print("Opção inválida, veja as opções
disponíveis:")
    for item in lista:
        print(item)

while True:
    lista = ["CARRO", "NAVIO", "ÔNIBUS", "AVIÃO"]
    resposta = input("Informe um meio de
transporte: ")
    if valida_elementos(resposta, lista):
        break
```

Vamos validar a entrada do usuário com base em uma lista fornecida verificando se o texto informado está contido na lista

MÓDULOS

- À medida que um programa aumenta e são adicionadas novas funções, fica impossível mantê-las em um único arquivo.
- É neste momento que entra o conceito de módulos.
- Todo arquivo **.py** é um módulo.
- Um programa pode conter diversos módulos.
- Um módulo pode usar outros módulos, desta forma aproveitamos códigos existentes.

EXEMPLO

operacoes.py

```
def media(*lista_numeros):
    total = 0
    for numero in
lista_numeros:
        total += numero
    resultado = total /
len(lista_numeros)
    return resultado
def soma(*lista_numeros):
    total = 0
    for numero in
lista_numeros:
        total += numero
    return total
```

programa.py

```
import operacoes

lista = []

while True:
    numero = int(input("Informe um número
int ou 0 para sair: "))
    if numero == 0:
        break
    lista.append(numero)
soma = operacoes.soma(*lista)
media = operacoes.media(*lista)
print(f"Soma: {soma}")
print(f"Média: {media}")
```

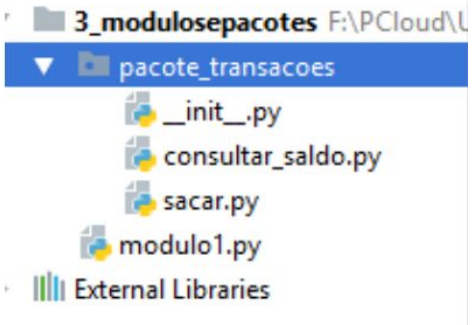
PACOTES

- Enquanto módulos são organizados em arquivos, pacotes ou packages são organizados em pastas identificadas com o arquivo __init__.py.
- Os pacotes funcionam como coleções para organizar módulos de forma hierárquica.
- É possível importar todos os módulos de um pacote, tirando os que começam com “__” (2 underscores) usando a declaração a seguir:

```
from nome_do_pacote import *
```

ARQUIVO _INIT_.PY

- O arquivo __init__.py indica ao Python que aquela pasta é um pacote importável. ode estar vazio, conter código de inicialização do pacote ou definir uma variável chamada __all__, que é uma lista de módulos que pertencem ao pacote e que serão importados quando usamos *.




```
__init__.py
__all__ =
['consultar_saldo','sac
ar']
```

IMPORTANDO PACOTES

O código abaixo importará todos os módulos de math, para importar apenas o necessário utilizamos from.

```
import math
print(math.sqrt(25))
```

O código abaixo importará o módulo sqrt do pacote math.

```
from math import sqrt
print(sqrt(25))
```

Observe que ao utilizar from package import item, o item pode ser um subpacote, submódulo, classe, função ou variável.

EXEMPLO

Vamos criar um pacote de módulos com a seguinte estrutura da pasta:

```
projeto/
├── programa.py
└── meu_pacote/
    ├── __init__.py
    ├── operacoes.py
    └── mensagens.py
```

📄 operacoes.py

```
def soma(a, b):
    return a + b
```

📄 mensagens.py

```
def ola():
    return "Olá, eu sou um módulo do pacote!"
```

EXEMPLO

📄 __init__.py
```
__all__ = ["operacoes", "mensagens"]
```

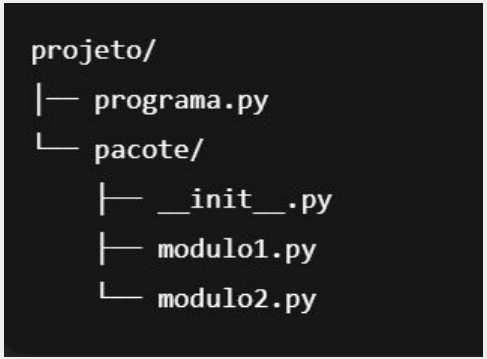
📄 programa.py (arquivo principal)
```
from meu_pacote import *

print(operacoes.soma(2, 3))
print(mensagens.ola())
```

<u>Resultado esperado:</u>

```
Olá, eu sou um módulo do pacote!
```

Sempre que estiver usando pacotes em Python, lembre:



👉 O arquivo que **executa** o programa deve ficar **fora do pacote**,
👉 e o pacote deve ter **init.py** para ser reconhecido pelo Python.

PACOTES - RESUMO DO EXEMPLO

Termo	Significa	Exemplo
Módulo	Um arquivo `.py`	`operacoes.py`
Pacote	Uma pasta com vários módulos e um `__init__.py`	`meu_pacote/`
`__init__.py`	Torna a pasta um pacote e controla o que será importado	`__all__ = [...]`
`from pacote import *`	Importa todos os módulos listados no `__all__`	`operacoes`, `mensagens`

EXERCÍCIOS

1. Faça um programa, com uma função interna que necessite de um argumento do tipo numérico decimal. A função retorna o valor de caractere ‘P’, se seu argumento for positivo, e ‘N’, se seu argumento for zero ou negativo.

2. Faça um programa com uma função IMPORTADA de um pacote externo chamada somaImposto. A função possui dois parâmetros formais: taxaImposto, que é a quantia de imposto sobre vendas expressa em porcentagem e custo, que é o custo de um item antes do imposto. A função “altera” o valor de custo para incluir o imposto sobre vendas.

EXERCÍCIOS

3. Faça um programa que utilize uma função chamada valorPagamento para calcular o valor devido de uma prestação. O programa deve solicitar ao usuário o valor da prestação e o número de dias de atraso, enviando essas informações para a função, que retornará o valor corrigido. O cálculo deve seguir estas regras: se não houver atraso, o valor permanece o mesmo; caso haja atraso, deve ser aplicada uma multa de 3% sobre o valor da prestação, além de juros de 0,1% ao dia. Após receber o valor calculado, o programa deve exibi-lo na tela e, em seguida, solicitar novamente um novo valor de prestação, repetindo o processo até que o usuário informe 0 como valor da prestação. Quando isso acontecer, o programa deve encerrar a execução e exibir um relatório com a quantidade de prestações pagas no dia e o total recebido.