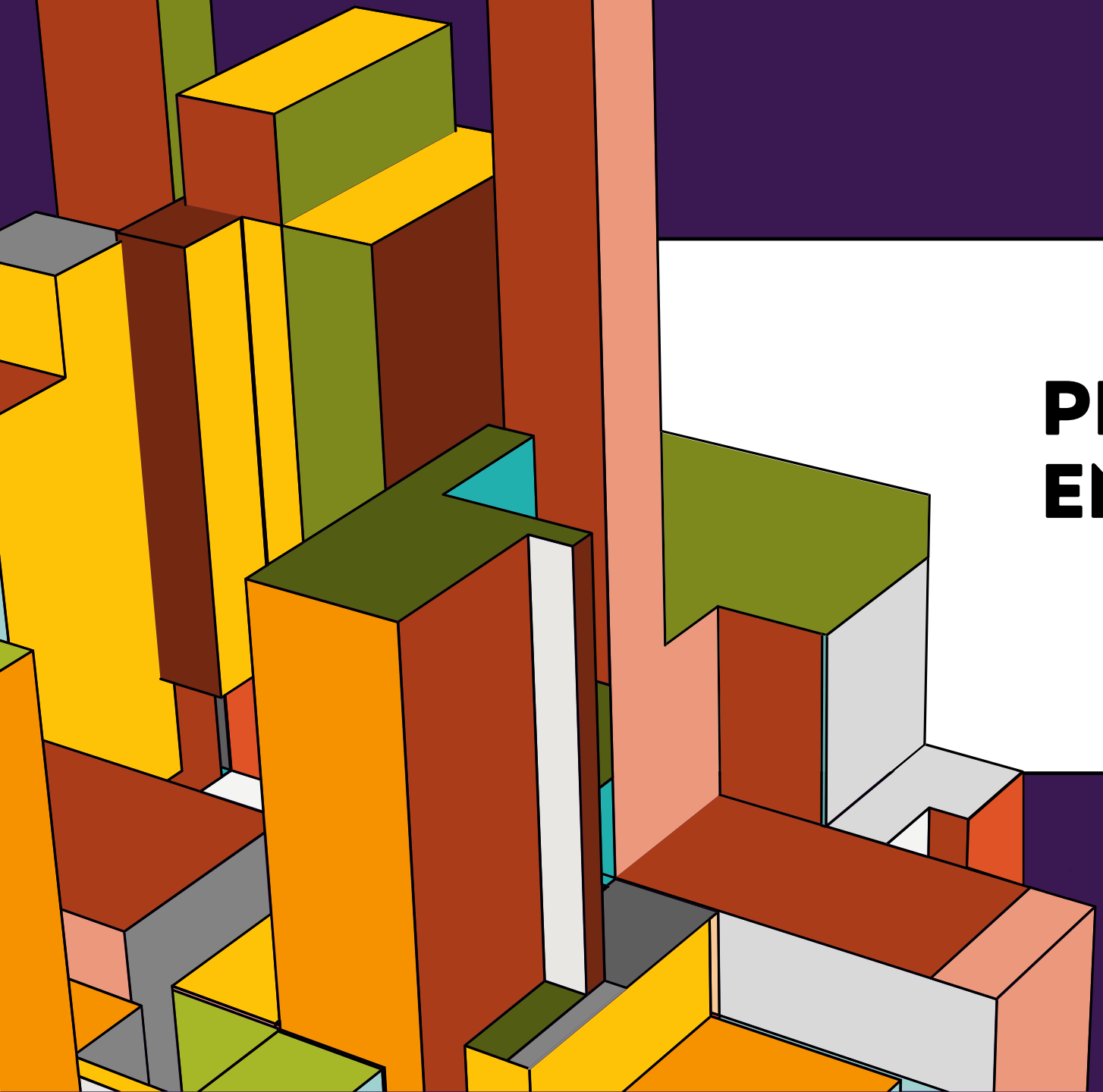


PROGRAMAÇÃO EM PYTHON

python

FUNÇÕES

De acordo com Downey (2015), uma função é uma sequência nomeada de instruções que executa uma determinada operação, sendo criada a partir de um nome e de instruções que a compõe, possibilitando ser utilizada posteriormente.

Funções são blocos de códigos que possuem um nome e podem ou não receber parâmetros.

Para criar uma função usamos a instrução `def`.

EXEMPLOS

```
def nomeDaFuncao(parametro):
    print('Esta função recebeu ', parametro, ' como parâmetro. ')
    print('O tipo deste parâmetro , é: ', type(parametro))
    return '\nCurso de Python é top!'
print(nomeDaFuncao('string'))
```

O bloco de código da função não é executado enquanto a mesma não for chamada

EXEMPLO 1

```
def soma(n1,n2):
    soma = n1+n2
    return soma
def subtracao(n1,n2):
    sub = n1-n2
    return sub
print('Bem vindo a calculadora')
num1=int(input('Digite o primeiro numero: '))
num2=int(input('Digite o segundo numero: '))
op = int(input('Digite 1 para somar e 2 para subtrair'))
if op == 1:
    print(f'O valor de soma {num1} e {num2} é {soma(num1,num2)}')
elif op ==2:
    print(f'O valor da subtração {num1} e {num2} é {subtracao(num1,num2)}')
else:
    print('Comando invalido!!')
```

UMA FUNÇÃO QUE CHAMA OUTRA FUNÇÃO

```
def fatorial(numero):
    if type(numero) is not int:
        return '\nO número deve ser um inteiro!'
    if numero <= 0:
        return 1
    else:
        return numero * fatorial(numero-1)
print(fatorial(8))
```

A partir do momento em que uma função é criada, é possível utilizá-la dentro de outras funções e, inclusive, dentro dela mesma.
Esse conceito é base para recursividade de funções. A seguir é mostrado um exemplo de uma função recursiva que calcula o fatorial de um número “n”.

MÉTODOS UTILIZADOS EM LISTAS

- Ao usarmos funções, começamos a trabalhar com variáveis locais e globais. Uma variável local, declarada dentro de uma função, existe apenas dentro desta função, sendo inicializada a cada chamada à função.
- Desta forma, ela não é visível fora da função. Uma variável global é definida fora de uma função, podendo ser vista por todas as funções do módulo e por todos os módulos que importam o módulo que a definiu.

VARIÁVEIS GLOBAIS E LOCAIS

Variáveis locais

Variáveis globais

```python
def soma(n1, n2):
    soma1 = n1 + n2
    return soma1
def subtracao(n1, n2):
    sub = n1 - n2
    return sub
print('Bem vindo a calculadora')
num1 = int(input('Digite o primeiro numero: '))
num2 = int(input('Digite o segundo numero: '))
op = int(input('Digite 1 para somar e 2 para subtrair'))
if op == 1:
    print(f'O valor de soma {num1} e {num2} é {soma(num1, num2)}')
elif op == 2:
    print(f'O valor da subtração {num1} e {num2} é {subtracao(num1, num2)}')
else:
    print('Comando invalido!!')
```

VARIÁVEIS GLOBAIS E LOCAIS

O escopo de nomes em Python é mantido por meio de namespaces, que são dicionários que relacionam os nomes dos objetos (referências) e os objetos em si.

Variáveis globais

```
valor = 100.00
def calculo():
    global valor
    valor = 50.00
    print(f"Valor dentro da função: {valor}")
print(f"Valor fora da função: {valor}")
calculo()
print(f"Valor fora da função: {valor}")
```

Os nomes ficam definidos em dois dicionários que podem ser consultados utilizando-se as funções locals() e globals(). Estes dicionários são atualizados em tempo de execução.

VARIÁVEIS GLOBAIS DEVEM SER UTILIZADAS O MÍNIMO POSSÍVEL, POIS DIFICULTAM A LEITURA E VIOLAM O ENCAPSULAMENTO DA FUNÇÃO (VEREMOS ENCAPSULAMENTO NA AULA DE ORIENTAÇÃO A OBJETOS).

UM BOM USO DE VARIÁVEIS GLOBAIS É UTILIZANDO AS MESMAS COMO CONSTANTES. SEMPRE QUE VÁRIAS FUNÇÕES PRECISEM DE UMA INFORMAÇÃO QUE É FIXA, PODEMOS CRIAR CONSTANTES GLOBAIS. NORMALMENTE CRIAMOS CONSTANTES EM MAIÚSCULAS.

PARÂMETROS DA FUNÇÃO

de funções são essenciais Parâmetros ou argumentos para que possamos chamar funções informando valores.

Desta forma, tornamos as funções flexíveis, recebendo valores diferentes a cada execução.

```
def soma():
    valor1 = 10
    valor2 = 25
    return valor1 + valor2

total = soma()
print(f"O total é: {total}")
```

FUNÇÃO COM PARÂMETROS OPCIONAIS.

valor1 e valor2 são obrigatórios, imprime é opcional porque já foi definida como falso.

```
def soma(valor1, valor2, imprime = False):
    resultado = valor1 + valor2
    if imprime:
        print(f"Soma: {resultado}")
    return resultado
total = soma(10, 84)
```

Parâmetros opcionais não podem estar no início caso seja combinado com parâmetros obrigatórios.

Total recebe 94, porém, nada é impresso, porque o valor padrão para “imprime” é falso e não foi informado na chamada.

NOMEANDO PARÂMETROS

```
def retangulo(largura, altura, caractere="*"):
    linha = caractere * largura
    for i in range(altura):
        print(linha)
retangulo(caractere="$", altura=5, largura=15)
```

Quando informamos o nome do parâmetro, não importa a ordem em que passamos os mesmos.

PARÂMETROS EMPACOTADOS EM LISTAS

```
lista = [10, 20, True]
def soma(valor1, valor2, imprime = False):
    resultado = valor1 + valor2
    if imprime:
        print(f"Soma: {resultado}")
    return valor1 + valor2
soma(lista[0], lista[1], True)
soma(*lista)
```

O empacotamento de lista evita termos que informar individualmente os valores da lista pelo índice.

O asterisco indica que estamos desempacotando a lista utilizando seus valores como parâmetros da função.

FUNÇÕES COMO PARÂMETROS

```
def soma(num1, num2):
    return num1 + num2
def multiplicacao(num1, num2):
    return num1 * num2
def calcular(funcao, num1, num2):
    return funcao(num1, num2)
total_soma = calcular(soma, 10, 20)
total_multiplicacao = calcular(multiplicacao, 10, 20)
print(f"Total Soma: {total_soma}")
print(f"Total Multiplicação: {total_multiplicacao}")
```

DESEMPACOTAMENTO DE PARÂMETROS

```
def soma(imprime, *valores):
    total = 0
    for valor in valores:
        total += valor
    if imprime:
        print(f"Soma: {total}")
    return total
soma(True, 10, 20, 30, 78)
soma(False, 10, 50)
```

Verifica-se que os valores enviados por lista são utilizados separadamente

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

```python
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
```

operacoes.py

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

programa.py

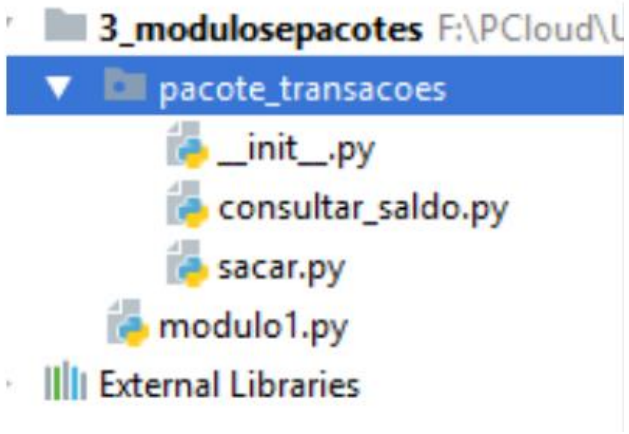
PACOTES

- 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 pode 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','sacar']
```

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.

EXERCÍCIOS

1. Faça um programa, com uma função que necessite de um argumento. 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 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 use a função **valorPagamento** para determinar o valor a ser pago por uma prestação de uma conta. O programa deverá solicitar ao usuário o valor da prestação e o número de dias em atraso e passar estes valores para a função **valorPagamento**, que calculará o valor a ser pago e devolverá este valor ao programa que a chamou. O programa deverá então exibir o valor a ser pago na tela. Após a execução o programa deverá voltar a pedir outro valor de prestação e assim continuar até que seja informado um valor igual a zero para a prestação. Neste momento o programa deverá ser encerrado, exibindo o relatório do dia, que conterá a quantidade e o valor total de prestações pagas no dia. O cálculo do valor a ser pago é feito da seguinte forma. Para pagamentos sem atraso, cobrar o valor da prestação. Quando houver atraso, cobrar 3% de multa, mais 0,1% de juros por dia de atraso.