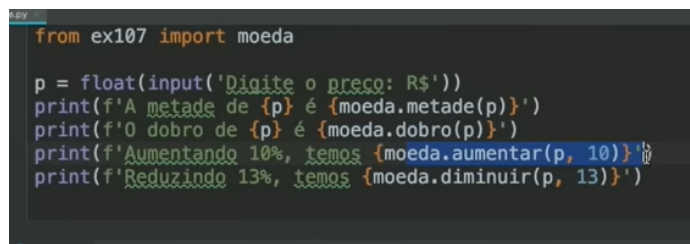


Mundo 3 - Módulos

1) Crie um módulo chamado moeda.py que tenha as funções incorporadas, `aumentar()`, `diminuir()`, `dobro()` e `metade()`. Faça também um programa que importe esse módulo e use algumas dessas funções.



```
from ex107 import moeda

p = float(input('Digite o preço: R$'))
print(f'A metade de {p} é {moeda.metade(p)}')
print(f'O dobro de {p} é {moeda.dobro(p)}')
print(f'Aumentando 10%, temos {moeda.aumentar(p, 10)}')
print(f'Reduzindo 13%, temos {moeda.diminuir(p, 13)}')
```

a)

Apenas um fato curioso sobre o Python, conseguimos no return utilizar calculos e fstrings

moeda.py

```
def dobro(num):
    return num*2

def metade(num):
    return num/2

def aumentar(num,perc):
    res = num + (num*perc/100)
    return res

def diminuir(num,perc):
    res = num - (num*perc/100)
    return res
```

--

main.py

```
from ex107 import moeda

p = float(input('Digite o preço: R$ '))
print(f'A metade de {p} é {moeda.metade(p)}')
print(f'O dobro de {p} é {moeda.dobro(p)}')
print(f'Aumentando 10%, temos {moeda.aumentar(p, 10)}')
print(f'Reduzindo 13% temos {moeda.diminuir(p, 13)}')
```

2) Adapte o código anterior criando uma função adicional chamada moeda() que consiga mostrar os valores como um valor monetário formatado

a) Neste método acrescentamos uma função chamada moeda onde ela recebe 2 parâmetros sendo o valor e o formato da moeda, após isso ele printa com uma f strins usando o método replace, trocando os pontos pelas vírgulas

```
main

from exes import moeda

p = float(input('Digite o preço: R$ '))
print(f'A metade de {moeda.moeda(p)} é {moeda.moeda(moeda.metade(p))}')
print(f'O dobro de {moeda.moeda(p)} é {moeda.moeda(moeda.dobro(p))}')
print(f'Aumentando 10%, temos {moeda.moeda(moeda.aumentar(p, 10))}')
print(f'Reduzindo 13% temos {moeda.moeda(moeda.diminuir(p, 13))}')

-----
--

moedas

def dobro(num):
    return num * 2

def metade(num):
    return num / 2

def aumentar(num, perc):
    res = num + (num * perc / 100)
    return res

def diminuir(num, perc):
    res = num - (num * perc / 100)
    return res

def moeda(p=0, moeda='R$'):
    return f'{moeda}{p:>10.2f}'.replace('.', ',')
```

3) Modifique as funções que foram criadas no desafio 1 para que elas aceitem um parâmetro a mais, informando se o valor retornado por elas vai ser ou não

formatado pela função moeda() desenvolvida no desafio 2

a)

Formatei as funções com os docstrings, fora isso para que possamos determinar se o usuário vai formatar ou não, ao invés de utilizarmos uma condicional e aumentar o código, podemos simplificar utilizando um terceiro parâmetro chamado formato.

Se o mesmo for True então ele formata, senão, não.

Algo interessante é a forma que podemos mesclar uma condicional na return de duas formas:

1. return <valor> if not <parâmetro> else <valor 2 a ser retornado>
2. return <valor> if <parâmetro> is <valor> else <valor 2 a ser retornado>

```
moeda

def dobro(num, formato=False):
    """
    -> Serve para dobrar o valor informado
    :param num: valor a ser dobrado
    :param formato: se vai ser formatado como função moeda
    :return: dobro do numero
    """
    return num * 2 if not formato else moeda(num)

def metade(num, formato=False):
    """
    -> Serve para dividir o valor em 2
    :param num: valor a ser dividido
    :param formato: se vai ser formatado como a função moeda
    :return: metade do valor
    """
    return num / 2 if formato is False else moeda(num/2)

def aumentar(num, perc, formato=False):
    """
    -> Serve para aumentar um valor em um percentual especificado
    :param num: valor a ser aumentado
    :param perc: percentual a ser aplicado de aumento
    :param formato: se vai ser formatado como a função moeda
    :return: aumento do valor
    """
    res = num + (num * perc / 100)
    return res if formato is False else moeda(res)

def diminuir(num, perc, formato=False):
    """
```

```

-> Serve para diminuir um número em um percentual especificado
:param num: valor a ser diminuído
:param perc: percentual a ser aplicado
:param formato: se vai ser formatado conforme a função moeda
:return: valor diminuído
"""

res = num - (num * perc / 100)
return res if not formato else moeda(res)

def moeda(p=0, moeda='R$'):
    """
    -> Serve para formatar a moeda para 10 espaços com 2 casas decimais divididas com , e não .
    :param p: valor
    :param moeda: Estilo da moeda, senão vai ser R$
    :return: valor formatado
    """
    return f'{moeda}{p:>10.2f}'.replace('.', ',')

-----
-----

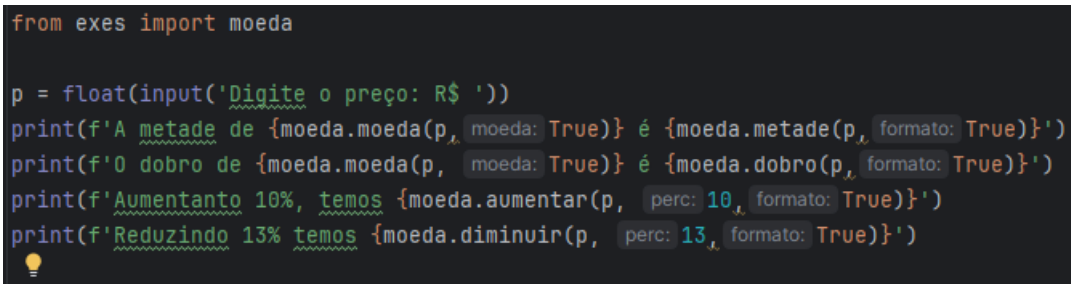
main

from exes import moeda

p = float(input('Digite o preço: R$ '))
print(f'A metade de {moeda.moeda(p, True)} é {moeda.metade(p, True)}')
print(f'O dobro de {moeda.moeda(p, True)} é {moeda.dobro(p, True)}')
print(f'Aumentando 10%, temos {moeda.aumentar(p, 10, True)}')
print(f'Reduzindo 13% temos {moeda.diminuir(p, 13, True)}')

```

O pycharm ajuda também no momento que designamos os parâmetros, então ele mostra estilo tags em qual parâmetro o valor na chamada da função vai entrar



```

from exes import moeda

p = float(input('Digite o preço: R$ '))
print(f'A metade de {moeda.moeda(p, moeda: True)} é {moeda.metade(p, formato: True)}')
print(f'O dobro de {moeda.moeda(p, moeda: True)} é {moeda.dobro(p, formato: True)}')
print(f'Aumentando 10%, temos {moeda.aumentar(p, perc: 10, formato: True)}')
print(f'Reduzindo 13% temos {moeda.diminuir(p, perc: 13, formato: True)}')

```

4) Adicione ao módulo moeda.py criado nos desafios anteriores uma função chamada `resumo()` que mostre na tela algumas informações geradas pelas funções que já temos no módulo criado até aqui.

a)

A forma que eu fiz foi criar uma função contendo todas as funções que utilizamos anteriormente, no final das funções faço os prints, então posso ver que realmente puxa as informações e conseguimos reduzir bastante os documentos. , mas o Guanabara utilizou outra forma.

Main

```
from exes import moeda
```

```
p = float(input('Digite o preço: R$ '))
moeda.resumo(p, True)
```

```
-----
-----
```

Moeda

```
def resumo(p, formato=False):
    def dobro(num, formato=False):
        """
        -> Serve para dobrar o valor informado
        :param num: valor a ser dobrado
        :param formato: se vai ser formatado como função moeda
        :return: dobro do numero
        """
        return num * 2 if not formato else moeda(num)

    def metade(num, formato=False):
        """
        -> Serve para dividir o valor em 2
        :param num: valor a ser dividido
        :param formato: se vai ser formatado como a função moeda
        :return: metade do valor
        """
        return num / 2 if formato is False else moeda(num/2)

    def aumentar(num, perc, formato=False):
        """
        -> Serve para aumentar um valor em um percentual especificado
        :param num: valor a ser aumentado
        :param perc: percentual a ser aplicado de aumento
        :param formato: se vai ser formatado como a função moeda
        :return: aumento do valor
        """
```

```

        res = num + (num * perc / 100)
        return res if formato is False else moeda(res)

def diminuir(num, perc, formato=False):
    """
    -> Serve para diminuir um número em um percentual especificado
    :param num: valor a ser diminuído
    :param perc: percentual a ser aplicado
    :param formato: se vai ser formatado conforme a função moeda
    :return: valor diminuído
    """
    res = num - (num * perc / 100)
    return res if not formato else moeda(res)

def moeda(p=0, moeda='R$'):
    """
    -> Serve para formatar a moeda para 10 espaços com 2 casas decimais
    divididas com , e não .
    :param p: valor
    :param moeda: Estilo da moeda, senão vai ser R$
    :return: valor formatado
    """
    return f'{moeda}{p:>10.2f}'.replace('.', ',')

print(f'A metade de {moeda(p, True)} é {metade(p, True)}')
print(f'O dobro de {moeda(p, True)} é {dobro(p, True)}')
print(f'Aumentando 10%, temos {aumentar(p, 10, True)}')
print(f'Reduzindo 13% temos {diminuir(p, 13, True)}')

```

Guanabara)

```

moedas

def dobro(num, formato=False):
    """
    -> Serve para dobrar o valor informado
    :param num: valor a ser dobrado
    :param formato: se vai ser formatado como função moeda
    :return: dobro do numero
    """
    return num * 2 if not formato else moeda(num)

def metade(num, formato=False):
    """
    -> Serve para dividir o valor em 2
    :param num: valor a ser dividido
    :param formato: se vai ser formatado como a função moeda
    """

```

```

        :return: metade do valor
        """
        return num / 2 if formato is False else moeda(num/2)

def aumentar(num, perc, formato=False):
    """
    -> Serve para aumentar um valor em um percentual especificado
    :param num: valor a ser aumentado
    :param perc: percentual a ser aplicado de aumento
    :param formato: se vai ser formatado como a função moeda
    :return: aumento do valor
    """
    res = num + (num * perc / 100)
    return res if formato is False else moeda(res)

def diminuir(num, perc, formato=False):
    """
    -> Serve para diminuir um número em um percentual especificado
    :param num: valor a ser diminuído
    :param perc: percentual a ser aplicado
    :param formato: se vai ser formatado conforme a função moeda
    :return: valor diminuído
    """
    res = num - (num * perc / 100)
    return res if not formato else moeda(res)

def moeda(p=0, moeda='R$'):
    """
    -> Serve para formatar a moeda para 10 espaços com 2 casas decimais divididas com , e não .
    :param p: valor
    :param moeda: Estilo da moeda, senão vai ser R$
    :return: valor formatado
    """
    return f'{moeda}{p:>.2f}'.replace('.', ',')

def resumo(p=0, taxa_a=0, taxa_d=0, formato=False):
    print('-'*30)
    print('RESUMO DO VALOR'.center(30))
    print('-'*30)
    print(f'Preço analisado: \t{moeda(p)}')
    print(f'Dobro do preço: \t{dobro(p, formato)}')
    print(f'Metade do preço: \t{metade(p, formato)}')
    print(f'{taxa_a}% de aumento: \t{aumentar(p, taxa_a, formato)}')
    print(f'{taxa_d}% de redução: \t{diminuir(p, taxa_d, formato)}')

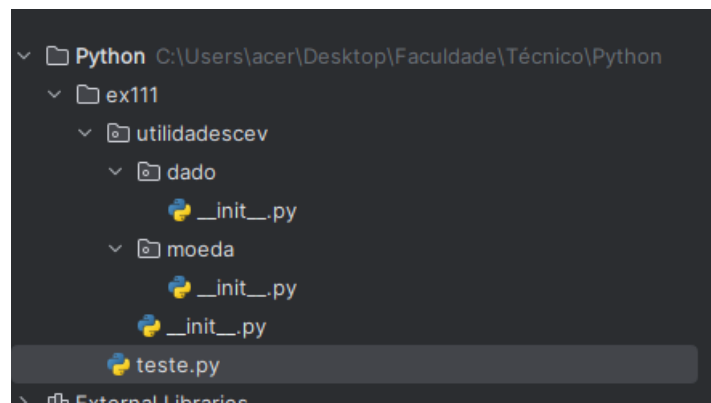
```

```
from exes import moeda

p = float(input('Digite o preço: R$ '))
moeda.resumo(p, 20, 30, True)
```

5) Crie um pacote chamado utilidadesCEV que tenha dois módulos internos chamados moeda e dado. Transfira todas as funções dos exercícios 1,2,3 para o primeiro pacote e mantenha tudo funcionando

Este exercício na verdade não tem muita coisa de código em si, mas mais de organização, então basicamente inserimos um pacote chamado utilidadescev, onde dentro dele existem 2 pacotes um moeda e um dado, após isso trouxe todas funções para moeda e importei para a main



Para importar utilizei o seguinte comando:

```
from ex111.utilidadescev.moeda import *
```

6) Dentro do pacote utilidadesCEV que criamos no desafio 5, temos um módulo chamado dado. Crie uma função chamada leiaDinheiro() que seja capaz de funcionar como a função input(), mas com uma validação de dados para aceitar apenas valores que sejam monetários.

Primeiramente no init do dado, foi criado o seguinte código

```
def leiaDinheiro(msg):
    valido = False
    while not valido:
        entrada = input(msg).replace('.', ',').strip()
        if entrada.isalpha() or entrada == '':
```



```

        print(f'\033[0;31mERRO! \'{entrada}\'' é um preço inválido\033
[m'])
    else:
        valido = True
        return float(entrada)

```

Entendendo o que ele faz

É uma estrutura bem simples, basicamente temos um validador, que avança para a próxima etapa se o valor dele for trocado por alguma condição, o professor tem o costume de usar esta estrutura, acho interessante colocar em prática em projetos futuros.

Lendo linha por linha:

1. declaramos a def leiaDinheiro recebendo apenas um parâmetro, sendo transportado para msg
2. declaramos a variável validadora chamada 'valido', inicialmente como False
3. enquanto não valido ou seja, enquanto valido é Falso, ele seguirá a estrutura
 - a. variável entrada recebe o input de msg, trocando o ponto final pela vírgula e dando um strip (isto é um exemplo de manipulação de dados)
 - b. se entrada for alpha ou entrada for vazio, então ele printa que o valor é inválido e volta para o input
 - c. senão valido recebe True e retorna para o global o valor Real de entrada

Já na parte global temos o seguinte

```

from ex111.utilidadescev import moeda
from ex111.utilidadescev import dado

p = dado.leiaDinheiro('Digite um Preço: R$ ')
moeda.resumo(p, 20, 30, True)

```

Entendendo o que está acontecendo

1. Basicamente estamos importando os dois diretórios como módulos, sendo o moeda e o dado
2. variável p recebe a função leiaDinheiro de dado onde é pedido para ele digitar um valor
3. abaixo está o resumo vindo da função moeda, onde estamos passando o valor em p que é o retornado da função anterior formatado como número real, os valores de acrescimo e decréscimo e se queremos que seja formatado ou não