Testes e TDD

1 Configurações iniciais

As configurações contidas nesse arquivo definem as configurações do vs code para compilador e para code runner

settings.json

```
{
    "editor.fontSize": 14,
    "editor.fontFamily": "Cascadia code",
    "editor.lineHeight": 20,
    "editor.quickSuggestions": {
        "other": true,
        "comments": false,
        "strings": true
    },
    "terminal.integrated.fontSize": 14,
    "terminal.integrated.fontFamily": "Cascadia Code",
    "[python]": {
        "diffEditor.ignoreTrimWhitespace": false,
        "editor.formatOnType": true,
        "editor.wordBasedSuggestions": "off",
        "editor.formatOnSave": true
    },
    "python.defaultInterpreterPath": "venv/bin/python3",
    "python.linting.flake8Enabled": true,
    "python.linting.mypyEnabled": true,
    "code-runner.executorMap": {
        "python": "clear && python3 -u"
    },
    "python.analysis.extraPaths": [
        "./src"
    ]
}
```

2 Assertions

Verifica se o caractere de entrada é válida.

assertions declaração e checagem

```
    assert isinstance(x, (int, float)), 'x precisa ser int ou float'
```

Uso no código

```
    def soma(x, y):
        assert isinstance(x, (int, float)), 'x precisa ser int ou float'
        assert isinstance(y, (int, float)), 'y precisa ser int ou float'
    return x + y
```

2.1 desativação de assertions

desativação shell

```
    python3 -O <nome_arquivo>
```

3 Testes via Doctests

3.1 Implementação

Listing:

```
    def subtrai(x, y):
    """
    >>> subtrai(30 ,  20)
        10
    >>> soma('a',  20)
    typerror

    """
    assert isinstance(x, (int, float)), 'x precisa ser int ou float'
    assert isinstance(y, (int, float)), 'y precisa ser int ou float'
    return x - y

    if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose= True)
```

4 Testes pelo unittest

teste para calculadora.py

```
try:
    import sys
    import os
    sys.path.append(
        os.path.abspath(
            os.path.join(
                os.path.dirname(__file__),
                '../src'
            )
        )
    )
except:
    raise
import unittest
from calculadora import soma

class TestCalculadora(unittest.TestCase):
    def test_soma_5_e_5_deve_retornar_10(self):
        self.assertEqual(soma(5, 5), 10)

    def test_soma_5_negativo_e_5_deve_retornar_0(self):
        self.assertEqual(soma(-5, 5), 0)

    def teste_soma_varias_entradas(self):
        x_y_saidas =(
            (10, 10, 20),
            (5, 5, 10),
            (1.5, 1.5, 3.0),
            (10, 10, 20),
            (10, 20, 30),

        )
        for x_y_saida in x_y_saidas:
            with self.subTest(x_y_saida = x_y_saida):
                    x, y, saida = x_y_saida
                    self.assertEqual(soma(x, y), saida)

    def soma_x_nao_int_ou_float_retorna_assertionerror(self):
        with self.assertRaises(AssertionError):
            soma(2, "a")

    def soma_y_nao_int_ou_float_retorna_assertionerror(self):
        with self.assertRaises(AssertionError):
            soma("a", 2)


if __name__ == "__main__":
    unittest.main(verbosity= 2)
```

5 Código final

main.py

```
from calculadora import soma


if __name__ =="__main__":
    x = 5
    y = 6

    """
    print(soma(x, y))
    print(soma(6, 7))
    print(soma(80, 56))
    print(soma(89, 90))
    """
    try:
        soma('15', 10)
    except TypeError as e:
        print(f"O erro foi {e}")
```

calculadora

```

def soma(x, y):
    """
    >>> soma(30 ,  20)
        50
    >>> soma('a',  20)
    """
    assert isinstance(x, (int, float)), 'x precisa ser int ou float'
    assert isinstance(y, (int, float)), 'y precisa ser int ou float'
    return x + y

if __name__ == "__main__":
```

```
    pass

def subtrai(x, y):
    """
    >>> subtrai(30 ,  20)
        10
    >>> soma('a',  20)
    typerror

    """
    assert isinstance(x, (int, float)), 'x precisa ser int ou float'
    assert isinstance(y, (int, float)), 'y precisa ser int ou float'
    return x - y

if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose= True)
```

6 TDD

Baconcomovos

```
"""
1. receber um numero inteiro
2. saber se o numero e multiplo de 3 e 5
bacon com ovos
3. saver se o numero multiplio somente de 3
bacon
3. saver se o numero multiplio somente de 5
ovos
2. saber se onumero nao e multiplo de 3 e 5
passa fome

"""

def bacon_com_ovos(n):
   assert(isinstance(n, int))
   if(n % 5 == 0 and n % 3 ==0):
      return "Bacon com ovos"
   elif(n%3 == 0):
      return "Bacon"
   elif(n%5==0):
      return "Ovos"
   return "Passa fome"
```

testeBaconcomovos

```
"""
TDD - Test Driven Development

Red
Parte 1 - criar o teste e ver falhar

Green
Parte2 - criar o codigo e ver oteste passar

refactor
Parte3 - melhorar meu codigo

"""
try:
    import sys
    import os
    sys.path.append(
        os.path.abspath(
            os.path.join(
                os.path.dirname(__file__),
                '../src'
            )
        )
    )
except:
    raise
import unittest
from baconcomovos import bacon_com_ovos

class TesteBaconComOvos(unittest.TestCase):
    def test_bacon_com_ovos_deve_levantar_assertion_error_se_nao_receber_int(self):
        with self.assertRaises(AssertionError):
           bacon_com_ovos(' ')

    def test_bacon_com_ovos_deve_retornar_bacon_com_ovos_se_entrada_for_multiplo_de_3_e_5(self):
      entradas = (15, 30, 45, 60)
      saida = "Bacon com ovos"
      for entrada in  entradas:
          self.assertEqual(
              bacon_com_ovos(entrada),
              saida,
              msg=f"{entrada} nao retornou {saida}"
```

```
            )

    def test_bacon_com_ovos_deve_retornar_passar_fome_se_entrada_nao_for_multiplo_de_3_e_5(self):
      entradas = (1,2,4, 7, 8)
      saida = "Passa fome"
      for entrada in  entradas:
          self.assertEqual(
              bacon_com_ovos(entrada),
              saida,
              msg=f"{entrada} nao retornou {saida}"
              )


    def test_bacon_com_ovos_deve_retornar_bacon_se_entrada_nao_for_multiplo_de_3(self):
      entradas = (3, 6, 9, 12, 18)
      saida = "Bacon"
      for entrada in  entradas:
          self.assertEqual(
              bacon_com_ovos(entrada),
              saida,
              msg=f"{entrada} nao retornou {saida}"
              )

    def test_bacon_com_ovos_deve_retornar_ovos_se_entrada_nao_for_multiplo_de_5(self):
      entradas = (5, 10, 20, 25)
      saida = "Ovos"
      for entrada in  entradas:
          self.assertEqual(
              bacon_com_ovos(entrada),
              saida,
              msg=f"{entrada} nao retornou {saida}"
              )




if __name__ =="__main__":
   unittest.main(verbosity = 2)
```

7 TDD pessoa.py

link para a documenetção unittest documentação

testePessoa.py

```
"""
class Pessoa:
    __init__
        nome str
        sobrenome str
        dados_obtidos bool

    API:
        obter_todos_os_dados -> method
            ok
            404
            (dados_obtidos se torna True se dados_obtidos com sucesso)
"""
try:
    import sys
    import os
    sys.path.append(
        os.path.abspath(
            os.path.join(
                os.path.dirname(__file__),
                '../src'
            )
        )
    )
except:
    raise

import unittest
from unittest.mock import patch
from pessoa import Pessoa

class TestPessoa(unittest.TestCase):
    def setUp(self):
        self.p1 = Pessoa('Luiz','Otavio')
        self.p2 = Pessoa('Marcos', 'Tome')

    def test_pessoa_attr_nome_tem_valor_correto(self):
        self.assertEqual(self.p1.nome , 'Luiz')
        self.assertEqual(self.p2.nome , 'Marcos')

    def test_pessoa_attr_nome_e_str(self):
        self.assertIsInstance(self.p1.nome , str)
        self.assertIsInstance(self.p2.nome , str)

```

```
    def test_pessoa_attr_sobrenome_tem_valor_correto(self):
        self.assertEqual(self.p1.sobrenome , 'Otavio')
        self.assertEqual(self.p2.sobrenome , 'Tome')

    def test_pessoa_attr_sobrenome_e_str(self):
        self.assertIsInstance(self.p1.sobrenome , str)
        self.assertIsInstance(self.p2.sobrenome , str)

    def test_pessoa_attr_dados_obtidos_inicia_false(self):
        self.assertFalse(self.p1.dados_obtidos)
        self.assertFalse(self.p2.dados_obtidos)

    def test_obter_todos_os_dados_sucesso_ok(self):
        with patch('requests.get') as fake_request:
            fake_request.return_value.ok = True
            self.assertEqual(self.p1.obter_todos_os_dados(), 'CONECTADO')
            self.assertTrue(self.p1.dados_obtidos)

        with patch('requests.get') as fake_request:
            fake_request.return_value.ok = True
            self.assertEqual(self.p2.obter_todos_os_dados(), 'CONECTADO')
            self.assertTrue(self.p2.dados_obtidos)

    def test_obter_todos_os_dados_falha_404(self):
        with patch('requests.get') as fake_request:
            fake_request.return_value.ok = False
            self.assertEqual(self.p1.obter_todos_os_dados(), 'ERRO 404')
            self.assertFalse(self.p1.dados_obtidos)

        with patch('requests.get') as fake_request:
            fake_request.return_value.ok = False
            self.assertEqual(self.p2.obter_todos_os_dados(), 'ERRO 404')
            self.assertFalse(self.p2.dados_obtidos)



    def test_obter_todos_os_dados_sucesso_e_falha_sequencial(self):
        with patch('requests.get') as fake_request:
            fake_request.return_value.ok = True

            self.assertEqual(self.p1.obter_todos_os_dados(), 'CONECTADO')
            self.assertTrue(self.p1.dados_obtidos)

            fake_request.return_value.ok = False
            self.assertEqual(self.p1.obter_todos_os_dados(), 'ERRO 404')
            self.assertTrue(self.p1.dados_obtidos)
        with patch('requests.get') as fake_request:
            fake_request.return_value.ok = True

            self.assertEqual(self.p2.obter_todos_os_dados(), 'CONECTADO')
            self.assertTrue(self.p2.dados_obtidos)

            fake_request.return_value.ok = False
            self.assertEqual(self.p2.obter_todos_os_dados(), 'ERRO 404')
            self.assertTrue(self.p2.dados_obtidos)


if __name__ =="__main__":
    unittest.main(verbosity = 2)
```

8 importando módulos para dentro do codigo

Rodando todos os testes **python3 -m unittest -v**

importação

```
    try:
    import sys
    import os
    sys.path.append(
        os.path.abspath(
            os.path.join(
                os.path.dirname(__file__),
                '../src'
            )
        )
    )
except:
    raise
```

9 type hints e mypy

rodando testes no arquivo especificado **mypy nomeArquivo**
A aula referente mypy do curso se referia a pycharm, e portanto as tipagens pareciam não funcionar

typehints.py

```
def add_number(n1:int ,  n2:float) -> int:
```

```
    if not n1:
        return 0

    return n1 + n2
initial: tuple[str, ...] = ("one", "two")

class Parent:
    def capitalise(self, text:str) -> str:
        return text.upper()

class child(Parent):
    def capitalise(self, text: str) -> str:
        return str(text).upper()
```