

5 Aula 05: 15/AGO/2019

5.1 Aula passada

Apresentação de um cliente e implementação parcial de uma classe `Complexo()`. Foram escritos os métodos:

- `__init__()`
- `__str__()`
- `__add__()`

Apresentação de um cliente e implementação parcial de uma classe `Polinomio`. Foram escritos os métodos.

- `__init__()`
- `__str__()`
- `__call__()`
- `derive()`

5.2 Hoje

Pilhas usando listas em Python, com os métodos `push()`, `pop()` e `isEmpty()`

5.3 Problema

Decidir se em um dada string de parênteses, colchetes e chaves está bem-formada.

Uma string de parênteses, colchetes e chaves é **bem-formada** se os parênteses, colchetes e chaves são fechados na ordem inversa àquela em que foram abertos.

Por exemplo, a primeira das sequências abaixo está bem-formada enquanto a segunda não está:

(() [()]) ([)]

5.4 Pilhas

Uma **pilha** (=stack) é uma lista dinâmica em que todas as operações:

- inserções: `push()`;
- remoções: `pop()`; e
- consultas: `peek()`.

são feitas em uma mesma extremidade chamada de **topo**.

```
empilha --->---+          +--->----> desempilha
(=push)      |          |          (=pop)
              v          |
              +-----+
              | vvvvvvvvv |
              +-----+
              | wwwwwwww |
              +-----+
              | zzzzzzzzz |
              +-----+
              | yyyyyyyyy |
              +-----+
              | xxxxxxxxx |
              +-----+
```

5.5 Solução “com classe” e dicionário

```
# Pilha.push(), Pilha.pop, Pilha.peek()
from stack import Stack

PROMPT = ">>> "
ABRES = "([{"
FECHAS = ")]}"
DICIO = {')' : '(', ']' : '[', '}' : '{'}
QUIT = ''

def main():
    """
    Recebe uma sequência formada apenas por parênteses e
    colchetes e verifica se é bem formada.
    """
    print("Verificador de sequencias bem formadas.")
    print("[Tecle ENTER para encerrar o programa.]")
    sequencia = input(PROMPT).strip()
    while sequencia != QUIT:
        if bem_formada(sequencia):
            print("bem-formada: sim")
        else:
            print("bem_formada: não")
        sequencia = input(PROMPT).strip()

#-----
def bem_formada(sequencia):
    """ (str) -> bool

    Recebe um string contendo uma sequencia de parênteses,
    chaves e colchetes e retorna TRUE se a sequência é bem
    formada e false em caso contrário.
    """
    pilha = Stack()
    for item in sequencia:
        if item in ABRES:
            pilha.push(item)
        elif item in FECHAS:
            if pilha.isEmpty():
                return False
            # verifique se o topo da pilha tem o abre certo
            item_topo = pilha.pop()
            if item_topo != DICIO[item]:
                return False

    if not pilha.isEmpty():
        return False
```

```
# passou pelos testes
return True

#-----
if __name__ == "__main__":
    main()
```

5.6 Solução “sem classe”

```
.....
# Usando push() e pop()

TESTE = False
PROMPT = "exp >>> "
ABRE_PARENTESES = "("
FECHA_PARENTESES = ")"
ABRE_CHAVES = "{"
FECHA_CHAVES = "}"
ABRE_COLCHETES = "["
FECHA_COLCHETES = "]"
ABRE = "([{"
FECHA = ")]}"
QUIT = ''

def main():
    '''
    Resolve um problema levemente mais geral.

    Recebe uma sequência de strings e para cada string verifica se a substring
formada apenas pelos seus parênteses, colchetes e chaves é bem formada.
    '''
    print("Verificador de sequencias bem formadas.")
    print("[Tecle ENTER para encerrar o programa.]")
    sequencia = input(PROMPT).strip()
    while sequencia != QUIT:
        if bem_formada(sequencia):
            print("bem-formada: sim")
        else:
            print("bem_formada: não")
        sequencia = input(PROMPT).strip()

#-----
def bem_formada(sequencia):
    ''' (str) -> bool

    Retorna True se a sequência é bem formada e False em caso contrário.
    '''
    pilha = []

    for item in sequencia:
        if item in [ABRE_PARENTESES, ABRE_COLCHETES, ABRE_CHAVES]: #
            pilha.push(item)
        elif item in [FECHA_PARENTESES, FECHA_COLCHETES, FECHA_CHAVES]:
            if pilha == []: # pilha vazia -- erro frequente
                return False
            # verifique se o topo da pilha tem o abre certo
```

```

        item_topo = pilha.pop()
        if item == FECHA_PARENTESES and item_topo != ABRE_PARENTESES:
            return False
        elif item == FECHA_COLCHETES and item_topo != ABRE_COLCHETES:
            return False
        elif item == FECHA_CHAVES and item_topo != ABRE_CHAVES:
            return False

    if len(pilha) > 0: ### pilha não vazia -- erro frequente
        return False

    # passou pelos testes
    return True

# outra versão que usa o modo index()
#-----
def bem_formada(sequencia):
    ''' (str) -> bool

    Recebe um string e verifica se a substring formada apenas por parênteses,
    chaves e colchetes da string é bem-formada.
    Retorna True se a sequência é bem formada e False em caso contrário.
    '''
    pilha = []

    for item in sequencia:
        if item in ABRE:
            pilha.push(item)
        elif item in FECHA:
            if pilha == []: # pilha vazia -- erro frequente
                return False
            # verifique se o topo da pilha tem o abre certo
            item_topo = pilha.pop()
            if ABRE.index(item_topo) != FECHA.index(item):
                return False

    if len(pilha) > 0: # pilha não vazia -- erro frequente
        return False

    # passou pelos testes
    return True

#-----
if __name__ == "__main__":
    main()

```

5.7 Classe Stack

```
class Stack:
    #-----
    def __init__(self):
        '''(Pilha) -> None

        Usado pelo construtor da classe.

        Monta um objeto da classe Pilha.
        '''
        self.itens = []

    #-----
    def __str__(self):
        '''(Pilha) -> str

        Recebe uma Pilha referenciada por `self` e constroi e
        retorna o string exibido por print() para imprimir uma
        pilha. Esse também é o string retornado por str().
        '''
        return str(self.itens)

    #-----
    def __len__(self):
        '''(Pilha) -> int

        Recebe uma Pilha referenciada por self e retorna
        o número de itens na pilha.

        Usado pelo Python quando escrevemos len(Pilha).
        '''
        return len(self.itens)

    #-----
    def isEmpty(self):
        '''(Pilha) -> bool

        Recebe uma Pilha referenciada por self e retorna
        True se ela está vazia e False em caso contrário.
        '''
        return self.itens == []

    #-----
    def push(self, item):
        '''(Pilha, objeto) -> None

        Recebe uma Pilha referenciada por self e um objeto
        item e coloca item no topo da pilha.
```

```

    '''
    self.itens.append(item)

#-----
def pop(self):
    '''(Pilha) -> objeto

    Recebe uma Pilha referenciada por self e desempilha
    e retorna o objeto no topo da pilha.
    '''
    return self.itens.pop()

#-----
def peek(self):
    '''(Pilha) -> objeto

    Recebe uma Pilha referenciada por self e retorna
    o objeto no topo da pilha. O objeto não é removido
    da pilha.
    '''
    return self.itens[-1]

```