Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES
Programação I / Programação Funcional
Prof. Oberlan Romão

Aula Prática 7 – Recursão

1) (ex1.py) Analise o código abaixo. Qual o objetivo de cada função? O que será impresso?

```
1  def f1(n, m):
2      if n <= m:
3          print(n)
4          f1(n + 1, m)
5  def f2(n, m):
6      if n > m:
7          return # mesmo que return None
8      else:
9          print(n)
10         f2(n + 1, m)
11 def f3(n, m):
12     print(n)
13     if n == m:
14         return None
15     f3(n + 1, m)
16 def f4(n, m):
17     print(n)
18     if n != m:
19         f4(n + 1, m)
20 def f5(n, m):
21     if n <= m:
22         f5(n + 1, m)
23         print(n)
24 def main():
25     print("f1(0, 10)")
26     f1(0, 10)
27     print("f2(0, 10)")
28     f2(0, 10)
29     print("f3(0, 10)")
30     f3(0, 10)
31     print("f4(0, 10)")
32     f4(0, 10)
33     print("f5(0, 10)")
34     f5(0, 10)
35
36 main()
```

Execute o arquivo `ex1.py`. Em seguida, modifique a função `f1` para que a mesma fique como apresentado abaixo e execute o código novamente (comente as chamadas das outras funções na `main()`). Você pode utilizar o site Python Tutor para facilitar a visualização da execução desse código.

```
1  def f1(n, m):
2      if n <= m:
3          print("Ida: ", n)
4          f1(n + 1, m)
5          print("Volta: ", n)
```

Observe que depois da chamada recursiva da função `f5` (linha 22), existe outra instrução a ser feita (`print(n)`). Essa instrução só é executada após o retorno da chamada recursiva. Coloque a instrução `print(n)` antes da chamada recursiva da função `f5` e execute novamente. A função gerou a mesma saída?

Em todas as chamadas das função, testamos valores tais que $n < m$. O que acontece, em cada função, se $n > m$? Volte no código e chame cada função passando o valor um valor de n maior que o valor de m. Quais funcionaram? Quais entraram em recursão infinita?

2) (`ex2.py`) Analise as funções recursivas abaixo e descubra qual o problema de cada uma. Como exemplo de execução, tente fazer (após a definição das funções) `print(f1(10))` e `print(f2(5))`.

```
1  def f1(n):
2      """Retorna a quantidade de números naturais pares menores ou iguais a n"""
3      if n % 2 == 0:
4          return 1 + f1(n - 2) #Por que n - 2?
5      else:
6          return f1(n - 1) #Por que n - 1?
7  def f2(n):
8      """Retorna a soma dos números naturais menores ou iguais a n"""
9      return 1 if n == 1 else n + f2(n + 1)
```

Corrija o(s) problema(s) de cada função. Por que uma das chamadas recursivas é `f1(n - 2)` e outra é `f1(n - 1)`?

3) (`ex3.py`) Analise e diga o que a relação de recorrência abaixo faz:

$$B(n) = \begin{cases} 0, & \text{se } n = 0 \\ n\%2 + 10 \times B(n//2), & \text{caso contrário} \end{cases}$$

Sugestão: implemente a relação de recorrência (transformando-a e uma função recursiva) e teste-a com diferentes valores para te ajudar a descobrir o objetivo da função. Por exemplo, `print(B(1))`, `print(B(2))`, `print(B(3))`, `print(B(4))` e assim por diante.

4) (`ex4.py`) Analise a função recursiva abaixo e tente descobrir o que ela faz. Faça o rastreio (teste de mesa) para `P(2)`, `P(3)` e `P(8)`. O que representa o parâmetro `nd`? Reescreva a função sem usar esse parâmetro.

```python
def P(n, nd = 0, i = 1):
    if n <= 1:
        return False
    if i == n:
        nd += 1
        return True if nd == 2 else False
    else:
        if n % i == 0:
            return P(n, nd + 1, i + 1)
        else:
            return P(n, nd, i + 1)
```

5) (ex5.py) Analise a função recursiva abaixo e tente descobrir o que ela faz. Faça o rastreio (teste de mesa) para `b(10)`.

```python
def b(n):
    if type(n) != int or n < 0: #n deve ser um número natural
        print("ERRO: O argumento deve ser inteiro positivo")
        return None
    elif n < 2: #Você consegue pensar em uma forma de simplificar essa parte?
        if n == 0:
            return "0"
        else:
            return "1"
    else: #Você consegue pensar em uma forma de simplificar o return?
         return b(n // 2) + ("0" if n % 2 == 0 else "1")
```

O que acontece se o último `return` for trocado por:
`return ("0"if n % 2 ==0 else "1")+ b(n //2)`?

É comum alguns programadores usarem `n % 1 == 0` para verificar se uma variável contém um número inteiro, ao invés de `type(n) == int` ou `isinstance(x, int)` (como feito no código anterior). Qual o problema de se usar `n % 1 == 0`?

6) (ex6.py) Analise as funções abaixo.

```python
def M(a, b):
    if not isinstance(b, int) or b < 0:
        print("ERRO")
        return None
    elif b == 0:
        return 0
    elif b == 1:
        return a
    else:
        return a + M(a, b - 1)


```

```python
def DI(a, b):
    if not isinstance(a, int) or not isinstance(b, int) or a < 0 or b <= 0:
        print("ERRO")
        return None
    if a < b:
        return 0
    else:
        return 1 + DI(a - b, b)

def RD(a, b):
    if a < b:
        return a
    else:
        return RD(a - b, b)
```

Qual o objetivo das funções `M(a, b)`, `DI(a, b)` e `RD(a, b)`? Qual será a saída quando for executado os comandos abaixo?

```python
print(M(2, 3))
print(M(4, 2))
print(DI(2, 3))
print(DI(3, 2))
print(DI(4, 2))
print(DI(10, 6))
print(RD(2, 3))
print(RD(3, 2))
print(RD(25.5, 10.5))
```

Criando módulos

Depois de criarmos várias funções, os programas ficam muito grandes e, muitas vezes, precisamos armazenar nossas funções em outros arquivos para, de alguma forma, usá-las, sem precisar reescrevê-las, ou pior, copiar e colar.

Em Python podemos resolver esse problema com módulos. Todo arquivos `.py` é um módulo, podendo ser importado com o comando `import <nomeDoArquivo>`, onde `<nomeDoArquivo>` (sem a extensão `.py`) representa o arquivo que contém a definição de todas as funções e constantes associadas ao módulo.

Coloque o arquivo `esfera.py` na pasta referente a essa aula prática. Abra o arquivo e analise as funções definidas. Crie um outro arquivo (`ex7.py`) e importe o módulo esfera. Em seguida, implemente uma função `main()` que utilize as três funções presentes no módulo.

Lembre-se que para usar uma função ou constante de um módulo precisamos fazer:
`<nomeModulo>.<nomeFucao>` ou `<nomeModulo>.<nomeConstante>`. Por exemplo:

```python
import math
print(math.sqrt(2))
```

ou

```
import math as m
print(m.sqrt(2))
```

Números aleatórios

Em alguns casos precisamos gerar números aleatórios ou randômicos para testar funções ou fazer simulações. Um número aleatório pode ser entendido como um número tirado ao acaso, sem qualquer ordem ou sequência predeterminada, como em um sorteio.

Para gerar número aleatórios em Python, podemos utilizar o modulo random. O módulo traz várias funções para geração de números aleatórios e mesmo números gerados com distribuições não uniformes. As principais funções do módulo são:

- random.random(): Gera um número aleatório x tal que $x \in [0, 1)$, ou seja, $0.0 \leq x < 1.0$.

  Exemplo:

```
import random
x = random.random() # 0 <= x < 1
print(x)
```

- random.uniform(a, b) Gera um número aleatório x tal que $x \in [a, b]$, ou seja, $a \leq x \leq b$.

  Exemplo:

```
import random
x = random.uniform(2.5, 10.0) # 2.5 <= x <= 10.0
print(x)
```

- random.randrange(a): Gera um número inteiro x tal que $x \in [0, a)$, ou seja, $0 \leq x \leq a-1$. Podemos definir o valor inicial do sorteio usando o comando random.randrange(a, b), nesse caso, $x \in [a, b)$.

  Exemplo:

```
import random
x = random.randrange(10) # 0 <= x <= 9
print(x)
x = random.randrange(5, 10) # 5 <= x <= 9
print(x)
```

- random.randint(a, b): Gera um número inteiro x tal que $x \in [a, b]$, ou seja, $0 \leq x \leq b$. Equivalente a random.randrange(a, b+1).

  Exemplo:

```
import random
x = random.randint(1, 6) # 1 <= x <= 6
print(x)
```

Exercícios:

1) Abra o arquivo `aleatorio1.py` e analise o código. Em seguida, execute-o diversas vezes. Os números sorteados foram os mesmos?

2) Analise as funções presentes no arquivo `aleatorio2.py`. Note a diferença entre `simulacaoV1` e `simulacaoV2`. Um jogador tem, em média, $\frac{1}{6} \approx 0.16667$ de chances de acertar. Foi encontrado uma média próxima de 0.16667?

3) (`ex8.py`) Faça uma função, `megaSena`, que receba 6 números (inteiros entre 1 e 60) apostado por um jogador na Mega-Sena. A função deve sortear outros 6 números (inteiros entre 1 e 60) e retornar a quantidade de números que o jogador acertou. Por exemplo, considerando que um jogador apostou nos números 1, 59, 28, 13, 7, 44 e foi sorteado 59, 10, 42, 1, 27, 33, a função deve retornar 2 (o jogador acertou os números 1 e 59). Note que a ordem da aposta e do sorteio não importam. Faça uma função `main()` que leia os 6 números apostados e, usando a função `megaSena`, imprima quantos números o jogador acertou.

Desafio: implemente uma função que também receba os 6 números apostados, faça a simulação de n ($1 \leq n \leq 1000$) sorteios e retorne: (i) a menor quantidade de números acertado entre as n simulações; (ii) a maior quantidade de acertos e; (iii) a média de acertos do jogador (o código de `aleatorio2.py` pode te ajudar). Em algum dos n sorteios o jogador acertou os 6 números? Qual foi a média de acertos?