Departamento de Ciência de Computadores — FCUP

Introdução à Programação I (CC1024) — 2021/2022

Folha 4

A menos que seja dito algo diferente no enunciado, deve implementar e testar as funções pedidas. Se o ficheiro em que as escrever for `prog_f4.py` e estiver no diretório corrente, lance o interpretador de Python e execute o comando `exec(open("prog_f4.py").read())` para **carregar as definições das funções no interpretador.**

1. Defina uma função `abs(x)` que retorna o valor absoluto de um inteiro `x`, passado como argumento.

```
>>> exec(open("prog_f4.py").read())
>>> abs(-100)
100
>>> abs(15)
15
>>> r = abs(7)
>>> r
7
```

2. Defina uma função `grau_rad(x)` para converter um valor `x` de graus para radianos (recorde que 180 graus correspondem a π radianos). Acrescente `import math` no início do ficheiro `prog_f4.py` para poder usar a constante π, definida no módulo `math` (para a referir usa `math.pi`). Depois de alterar o ficheiro, tem de voltar a recarregar o programa.

```
>>> exec(open("prog_f4.py").read())
>>> k = grau_rad(360)
>>> k
6.283185307179586
>>> grau_rad(45)
0.7853981633974483
>>> print(grau_rad(180))
3.141592653589793
```

3. Defina uma função `posicao(a,b,c,d)` que, dadas as coordenadas `(a,b)` e `(c,d)` de dois pontos P e Q do plano, expressas no referencial cartesiano usual, carateriza a posição de Q relativamente a P, usando `esquerda`, `direita`, `acima` ou `abaixo`. Se coincidirem, indica `coincidente`. A função retorna uma *string*. Para facilitar a estruturação do código, note que a construção de uma resposta composta `resp`, como `'direita abaixo'` ou `'direita acima'` pode ser efetuada por concatenação, de `'direita'` com `'abaixo'` ou `'acima'`, usando o operador $+$. Inicialmente, podemos inicializar `resp` com a *string* vazia (`resp = ''`).

```
>>> exec(open("prog_f4.py").read())
>>> posicao(-1,2,4,-1)
'direita abaixo'
>>> print(posicao(4,-1,-1,2))
esquerda acima
```

```
>>> posicao(4,-1,4,3)
'acima'
>>> posicao(4,-1,4,-1)
'coincidente'
>>> x = posicao(4,-1,5,-1)
>>> print(x)
direita
```

4. Na continuação de **3.**, escreva um procedimento `classifica()` que irá ler um inteiro n do *standard input* e a seguir n pares de pontos com coordenadas inteiras e os vai classificando. Para cada par (P, Q), com $P = (a, b)$ e $Q = (c, d)$, **lê uma linha com inteiros a, b, c e d separados por um espaço**, e escreve no *standard output* uma mensagem no formato seguinte, com a, b, c, d e pos substituídos corretamente:

```
posição de Q=(c,d) relativamente a P=(a,b): pos
```

Escreva o programa num ficheiro `classpontos.py`, completando o código seguinte.

```
def posicao(a,b,c,d):
    "classificar posição de (c,d) relativamente a (a,b)"
    # completar

def classifica():
   "classificar a posição de relativa de varios pares de pontos"
    n = int(input())
    # completar

classifica()   # chamar a função que vai ser executada
```

A seguir, escreva os dados num ficheiro (por exemplo, `dados1`) e execute no terminal

`python classpontos.py < dados1 > res1`

para executar o programa com redirecionamento do *standard input* para `dados1` e do *standard output* para `res1`. Pode ver o conteúdo de `res1` se escrever `cat res1` no terminal (i.e., na *shell de comandos*).

Na função `classifica()`, a instrução para saída de *output* (imprimir a mensagem) pode ser:

```
print("posição de Q=(%d,%d) relativamente a P=(%d,%d): %s" % (c,d,a,b,pos))
```

se `pos` contiver a *string* (cadeia de caracteres) que indica a posição. Nesse formato, `%s` é usado para escrita de *strings*.

Sobre os exercícios 5 e 6:

Se tivermos uma sucessão $u_1, u_2, u_3, \ldots$, usamos

$$\sum_{i=1}^{n} u_i$$

para designar a soma dos n primeiros termos, isto é $S_n = u_1 + u_2 + \cdots + u_n$. Para a soma de todos os termos usamos $\sum_{i=1}^{\infty} u_i$, correspondendo ao limite de S_n quando n tende para infinito, isto é, $\lim_{n\to\infty} \sum_{i=1}^{n} u_i$, o qual pode existir ou não.

Por exemplo, $\sum_{i=1}^{\infty} \frac{1}{i} = 1 + (1/2) + (1/3) + \cdots$ não existe (é um infinitamente grande).

Mas, $\sum_{i=1}^{\infty} \frac{1}{2^{i-1}} = 1 + (1/2) + (1/4) + (1/8) + \cdots = \lim_{n\to\infty} \frac{1-(1/2)^n}{1-1/2}$ existe (é igual a 2).

Nas UCs de Cálculo I e Cálculo II irão ver que algumas funções, como $x \rightsquigarrow \sin(x)$, $x \rightsquigarrow \cos(x)$, $x \rightsquigarrow e^x$, podem ser definidas por séries, o que nos permite obter valores aproximados do valor da função num ponto x, usando a soma dos primeiros k termos da série, para um valor de k razoável. Teoricamente, quanto maior for o valor de k, melhor seria a aproximação. Como um computador efetua cálculos com precisão finita (nem todos os valores podem ser representados), as operações estão sujeitas a erros numéricos e o valor deixa de melhorar ainda que aumentemos k.

5. A fórmula de Leibniz para aproximar π é:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \cdots\right) = 4 \times \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Implemente a função `leibniz(k)` que calcula a somas dos primeiros `k` termos desta série. Não deve calcular $(-1)^n$ diretamente. Note que o valor alterna entre -1 e 1, pelo que, basta manter uma variável `sinal`, que em cada iteração troca de sinal, por execução da instrução `sinal = -sinal`.

6. Considere as séries

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \qquad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

a) Implemente funções `seno(x,k)` e `cosseno(x,k)` para calcular aproximações de $\sin(x)$ e de $\cos(x)$, considerando a soma dos primeiros `k` termos. Não deve usar funções para calcular fatoriais. Deve ver como obter o termo de ordem n a partir do termo de ordem $n-1$, para $n \geq 0$.

b) Teste as funções para alguns valores de `x` fazendo variar `k`. Compare os resultados com o valor calculado por `math.sin(x)` e `math.cos(x)` do módulo `math`.