Este documento apresenta uma revisão dos principais assuntos abordados em Programação 1.

Aula 1: Fundamentos da programação estruturada

Qual a diferença entre **programa**, **linguagem de programação**, **código fonte** e **agoritmo**?

Para o computador, um **programa** é apenas uma sequência de instruções (código de máquina) composto por 0's e 1's. Máquinas não entendem linguagens humanas (português, inglês, *etc*) e humanos, geralmente, não entendem código de máquina. Uma **linguagem de programação** é um meio termo entre humanos e máquinas. Possui um conjunto de regras para descrever um funcionamento de um programa. Um **código-fonte** contém **algoritmos** escritos em alguma linguagem de programação. É preciso haver uma **tradução** de código-fonte escrito em uma certa linguagem de programação para que ele seja transformado em um programa executável por um computador.

Um **paradigma de programação** define o "estilo" de uma linguagem de programação. Existem diversos paradigmas de programação diferentes. Exemplos: *paradigma estruturado*, *paradigma funcional*, *paradigma orientado a objetos*. Python é multiparadigma: é possível escrever código em Python usando o paradigma estruturado, funcional e OO. Entretanto, estudamos apenas o paradigma estruturado nesta disciplina. A programação estruturada é composta, basicamente, por:

- Sequências de instruções:

```
1 x = 10
2 y = 30
3 z = x+y
4 print(z)
```

  O computador irá sempre executar as instruções na ordem em que aparecem no código, a não ser que haja algum comando que o faça desviar para outro trecho do código.

- Entrada e saída (*Input/Output*, ou apenas I/O): Um programa pode interagir com o usuário de diversas formas. Existem dispositivos de entrada de dados, para que o usuário passe comandos ao programa (como *mouse* e teclado), e os dispositivos de saída, para que ele receba informações do programa (como o monitor, uma impressora ou até mesmo um HD). Dizemos que o teclado é a entrada de dados padrão (*standard input*) e o monitor é a saída padrão (*standard output*). Em Python, a entrada de dados padrão é feita pela função **input** e a saída padrão é feita pela função **print**:

```
1 # Lendo dois numeros e transformando-os em inteiros:
2 x = int(input("Digite um numero: "))
3 y = int(input("Digite outro numero: "))
4
5 # Imprimindo a soma dos dois numeros:
6 print(x, "+", y, "=", x+y)
7
8 # Outra formatacao possivel para a mesma impressao:
9 print("{} + {} = {}".format(x, y, x+y))
```

  Se os número digitados forem $x = 45$ e $y = 3$, por exemplo, o trecho de código acima irá imprimir na tela:

```
45 + 3 = 48
45 + 3 = 48
```

- **Decisões**: O comando **if** em Python desvia o código para trechos diferentes dependendo do resultado de uma condição, geralmente expressa por uma expressão booliana (verdadeiro ou falso):

```
1 if x>y:
2     print(z)
3 else:
4     print(x+y)
```

- **Repetições** (iterações): Repetições em Python podem ser realizadas com os comandos **while** ou **for**. O primeiro executa um trecho de código **enquanto** a condição for verdadeira. O segundo executa uma iteração **para cada** elemento em uma lista (veja mais sobre listas na próxima aula). Chamamos o bloco a ser repetido de laço (ou *loop*, em inglês):

```
1 while x<y:
2     print("Oi!")
3     x = x+1
4
```

```
5  frutas = ["pera", "uva", "maca"]
6  for fruta in frutas:
7      print(fruta)
8
9  for n in range(10):
10     # Lista com os 10 primeiros inteiros a partir de 0
11     print(n)
```

Cuidado: não confunda **interação** (quando o usuário troca informações com o programa que está em execução) com **iteração** (cada vez que um trecho de código é repetido).

- Sub-rotinas (funções, métodos ou procedimentos): blocos de comandos que realizam tarefas específicas e atendem a alguma necessidade, independente de quando ou qual trecho de código a solicitar. Uma **função** é uma sub-rotina que recebe um conjunto de parâmetros (que pode ser vazio), calcula um resultado e retorna-o para o trecho de código em que foi chamada. O fatorial de um número, por exemplo, é definido por:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

Matematicamente, podemos definir a função fatorial como:

$$n! = \begin{cases} n \times (n-1)!, & \text{se } n > 0 \\ 1, & \text{caso contrário.} \end{cases}$$

A linguagem Haskell (que utiliza o paradigma funcional), define funções de forma bem semelhante às definições matemáticas. A função fatorial em Haskell, por exemplo, seria:

```
1  fat 0 = 1
2  fat n = n * fat (n - 1)
```

Em Python, podemos definir a função de duas formas:

```
1  def fat(n):
2      if n > 0:
3          return n * fat(n-1)
4      else:
5          return 1
6
7  def fat2(n):
8      x = 1
```

```
9       while n > 0:
10          x = x * n
11          n = n - 1
12      return x
```

A primeira função, mais semelhante à definição matemática, é **recursiva** (ou seja, contém uma chamada à própria função fatorial). A segunda função é **iterativa**, ou seja, utiliza um comando de repetição (neste caso, um **while**) para acumular o resultado do cálculo de $n \times (n-1) \times (n-2) \times \cdots \times 1$. As duas funções recebem apenas um parâmetro (o número n) e retornam o fatorial desse número como resultado. O retorno de uma função é definido pelo comando **return**.

Uma função também pode fazer chamada à outra função e utilizar o valor retornado por ela. Uma combinação simples, por exemplo, é dada por:

$$C_{(n,p)} = \frac{n!}{p! \times (n-p)!}$$

Em Python, podemos defini-la como:

```
1 def comb(n, p):
2     num = fat(n)
3     den = fat(p) * fat(n-p)
4     return num / den
```

Quando fazemos chamada a alguma função, podemos salvar o seu valor de retorno em alguma variável (segunda linha da função **comb**), usá-lo diretamente em outra expressão (terceira linha da função **comb**) ou até mesmo imprimir diretamente o valor.

Um **procedimento** é uma sub-rotina que não possui um valor de retorno. Um procedimento em Python, assim como uma função, é definido pelo comando **def**, mas não possui um valor de retorno. Podemos, por exemplo, criar um procedimento que imprime os n primeiros múltiplos de k:

```
1 def multiplos(n, k):
2     i = 1
3     print("Os {} primeiros multiplos de {} sao:".format(n, k))
4     while i <= n:
5         print(i*k)
6         i = i+1
```

A saída do procedimento para $n = 10$ e $k = 3$ seria:

```
Os 10 primeiros multiplos de 3 sao:
3
```

```
6
9
12
15
18
21
24
27
30
```

De forma parecida, imprimimos a seguir todos os múltiplos de k até n:

```
1 def nMultiplos(n, k):
2     i = 1
3     print("Os multiplos de {} ate {} sao:".format(k, n))
4     while i*k <= n:
5         print(i*k)
6         i = i+1
```

A saída do procedimento para $n = 10$ e $k = 3$ seria:

```
Os multiplos de 3 menores ou iguais a 10 sao:
3
6
9
```

Note que, como um procedimento não possui um valor de retorno, a chamada de um procedimento não deve ser armazenada em uma variável. Exemplo:

```
1 # Retorno da funcao input sendo transformado para inteiro
2 # e depois sendo salvo na variavel x:
3 x = int(input("Digite um numero: "))
4
5 # Retorno da funcao fat sendo impresso diretamente:
6 print("{}! = {}".format(x, fat(x))
7
8 # Imprimindo os 5 primeiros multiplos de x:
9 multiplos(5, x)
```

Por fim, um **método** é uma sub-rotina que pertence aos objetos de uma classe em uma linguagem orientada a objetos (assunto de outra disciplina).

Exercícios Resolvidos da Aula 0: Fundamentos da programação estruturada

1. **Divisor**: Dados dois inteiros positivos x e y, verifique se x é divisor de y. O retorno deve ser booliano (verdadeiro ou falso).

```
1 def divisor(x, y):
2     if y%x == 0:
3         return True
4     else: return False
```

2. **Divisores**: Dado um número k, imprima todos os divisores de k.

```
1 from divisor import divisor
2
3 def divisores(k):
4     x = 1
5     while x <= k:
6         if divisor(x, k):
7             print(x, end=" ")
8         x = x+1
9     print()
```

3. **Máximo Divisor Comum (MDC)**: Dados dois números m e n, imprima o máximo divisor comum entre m e n.

```
1 from divisor import divisor
2
3 def maior(m, n):
4     if m > n:
5         return m
6     else:
7         return n
8
9 def mdc(m, n):
10    i = 1
11    maximo = 1
12    while i <= maior(m, n):
13        if divisor(i, m) and divisor(i, n):
14            maximo = i
15        i = i+1
16    print("MDC({},{}) = {}".format(m, n, maximo))
```

4. **Primo**: Dado um número x, verifique se ele é primo.

```
from divisor import divisor

def primo(x):
    i = 2
    divisores = 0
    while i < x:
        if divisor(i, x):
            divisores = divisores + 1
        i = i+1
    # x eh primo se ele tem zero divisores alem de 1 e ele mesmo
    return divisores == 0
```

5. **Primos**: Dado um número k, imprima todos os números primos até k.

```
from divisor import divisor
from primo import primo

def primos(k):
    i = 2
    while i <= k:
        if primo(i):
            print(i, end=" ")
        i = i+1
    print()
```

6. **Dama**: *[Maratona de Programação 2008]* O jogo de xadrez possui várias peças com movimentos curiosos. Uma delas é a dama, que pode se mover qualquer quantidade de casas na mesma linha, na mesma coluna, ou em uma das duas diagonais, conforme exemplifica a Figura 1.

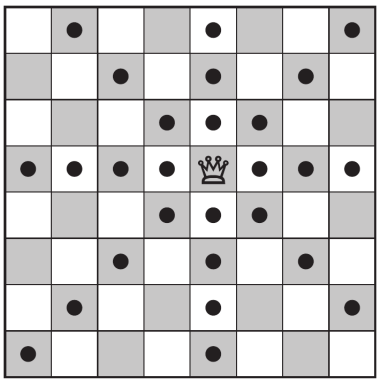

Figura 1: Movimentos possíveis da dama em um tabuleiro de xadrez.

Dada duas posições (x, y) e (m, n) em um tabuleiro de xadrez vazio (ou seja, um tabuleiro 8×8, com 64 casas), calcule e imprima a quantidade mínima de movimentos que a dama precisa fazer para ir da posição (x, y) para a posição (m, n).Exemplos:

Entrada		***Saída***
(x, y)	(m, n)	
$(4, 4)$	$(6, 2)$	1
$(3, 5)$	$(3, 5)$	0
$(5, 5)$	$(4, 3)$	2

```
1  def abs(x):
2      if x < 0: return -x
3      else: return x
4
5  def mesmaPos(x1, y1, x2, y2):
6      return x1 == x2 and y1 == y2
7
8  def alinhados(x1, y1, x2, y2):
9      return x1 == x2 or y1 == y2 or (abs(x1-x2) == abs(y1-y2))
10
11 def dama(x, y, m, n):
12     if (mesmaPos (x, y, m, n)): print(0)
13     elif (alinhados (x, y, m, n)): print(1)
14     else: print(2)
```

7. **Acerola**: *[Maratona de Programação 2008]* Natural das Antilhas, a acerola (Malpighia glabra Linn, também conhecida como cereja das Antilhas) já era apreciada pelos nativos das Américas há muitos séculos. Mas o grande interesse por essa fruta surgiu na década de 1940, quando cientistas porto-riquenhos descobriram que a acerola contém grande quantidade de ácido ascórbico (vitamina C). A acerola apresenta, em uma mesma quantidade de polpa, até 100 vezes mais vitamina C do que a laranja e o limão, 20 vezes mais do que a goiaba e 10 vezes mais do que o caju e a amora.

   Um grupo de amigos está visitando o Sítio do Picapau Amarelo, renomado produtor de acerola. Com a permissão de Dona Benta, dona do sítio, colheram uma boa quantidade de frutas, e pretendem agora fazer suco de acerola, que será dividido igualmente entre os amigos durante o lanche da tarde.

   Conhecendo o número de amigos, a quantidade de frutas colhidas, e sabendo que cada unidade da fruta é suficiente para produzir 50 ml de suco, escreva uma função que receba como parâmetros o número N de amigos e a quantidade F de frutas

colhidas, e imprima com precisão de duas casas decimais qual o volume, em litros, que cada amigo poderá tomar. Exemplos:

Entrada		***Saída***
N	*F*	
1	1	0.05
5	431	4.31
101	330	0.16

```
1 def acerola(N, F):
2     q = F*0.05
3     print("%.2f"%(q/N))
```

8. **Alarme Despertador**: *[Maratona de Programação 2009]* Daniela é enfermeira em um grande hospital e tem os horários de trabalho muito variáveis. Para piorar, ela tem sono pesado, e uma grande dificuldade para acordar com relógios despertadores. Recentemente ela ganhou de presente um relógio digital, com alarme com vários tons, e tem esperança que isso resolva o seu problema. No entanto, ela anda muito cansada e quer aproveitar cada momento de descanso. Por isso, carrega seu relógio digital despertador para todos os lugares, e sempre que tem um tempo de descanso procura dormir, programando o alarme despertador para a hora em que tem que acordar.

   No entanto, com tanta ansiedade para dormir, acaba tendo dificuldades para adormecer e aproveitar o descanso. Um problema que a tem atormentado na hora de dormir é saber quantos minutos ela teria de sono se adormecesse imediatamente e acordasse somente quando o despertador tocasse. Mas ela realmente não é muito boa com números, e pediu sua ajuda para escrever uma função que, dada a hora corrente e a hora do alarme, determine o número de minutos que ela poderia dormir. Exemplos:

Entrada				***Saída***
Hora atual	***Minuto atual***	***Hora alarme***	***Minuto alarme***	
1	5	3	5	120
23	59	0	34	35
21	33	21	10	1417

```
1 def alarme(hora_atual, minuto_atual, hora_alarme, minuto_alarme)
2     t1 = hora_atual*60 + minuto_atual
3     t2 = hora_alarme*60 + minuto_alarme
4
5     if (t1 < t2):
```

```
6           # Dorme e acorda no mesmo dia:
7           print(t2-t1)
8       else:
9           # Dorme e acorda em dias diferentes
10          print (24*60 + t2-t1)
```