

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

Aula 0: Fundamentos da programação estruturada

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



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 $n \times (n-1) \times (n-2) \times \dots \times 1$. 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).