

# IF669 - Introdução a Programação

Marcelo Anderson Oliveira de Santana

Dezembro, 2021

## 1 Introdução

### 1.1 Sobre a disciplina

O projeto aborda sobre a disciplina “Introdução a Programação”, ministrada no curso de graduação de Ciência da Computação, da Universidade Federal de Pernambuco. Como forma de desenvolver o raciocínio lógico e introduzir os alunos à programação, a disciplina adota *Python* como linguagem utilizada, sendo a mesma de tipagem dinâmica. Além do mais, 'IP' possui uma carga horária total de 120h, é obrigatória na grade curricular do discente e possui crédito 6.

### 1.2 Sobre a Metodologia de Ensino

Sobre a metodologia de ensino, três professores ministram a disciplina, revezando entre si. O sistema de notas é dividido em: 70% são listas de treinamento e 30% correspondem ao miniprojeto no final do semestre. Têm-se duas plataformas de estudos: o *Dikastis*, sendo o ambiente onde são postadas as listas de treinamento que compõem a nota, e o *Redu*, plataforma na qual tem o conteúdo semanal, em forma de vídeo e por cadernos de *Jupyter Notebook*.

A forma de contato se dá principalmente pelos horários de aula síncronas, nas quais os monitores tiram dúvidas dos estudantes, e também pelo servidor do Discord, onde também existe meio de comunicação entre os discentes e os professores.

## 2 Conteúdos

A forma de realizar um algoritmo dentro das diferentes linguagens de programação é o que faz com que elas sejam diferentes entre si, ou seja, a sintaxe de uma linguagem para outra é diferente, mas o que se quer resolver, não. Por isso, durante todo o semestre da disciplina, são estudadas estruturas gerais, presentes em grande parte das linguagens.

O que diferencia essas linguagens é a maneira de escrever o código para execução de algum comando específico, sendo esse comando, em sua grande maioria, presente em outra linguagem, diferindo apenas na maneira de utilizá-lo.

Abaixo, seguem os comandos abordados durante o semestre, juntamente com uma breve explicação do que os mesmos podem realizar.

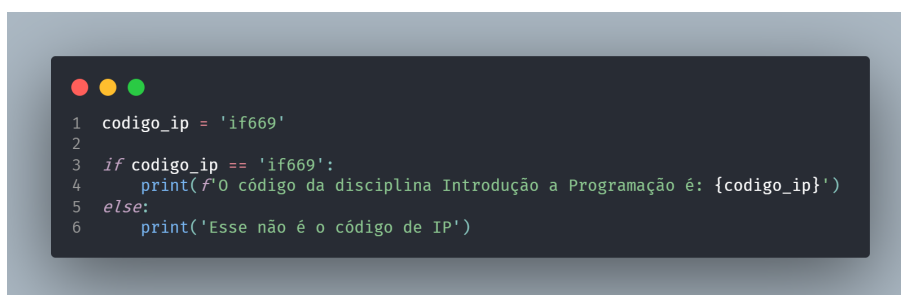
## 2.1 Comandos Condicionais

Os comandos condicionais são utilizados quando queremos analisar uma certa expressão e, a partir desse resultado, executar certo bloco de código que está indentado nessa condição.

Em *Python*, para estruturas condicionais, têm-se o *if*, *elif* e *else*. O *if*, como a própria tradução do Inglês já diz, executa aquele código **se** tal proposição for verdadeira.

Em alguns casos, temos várias condições dependentes, e são nesses casos que usamos o *elif*: quando uma primeira proposição não é verdadeira, a elaboração de outra expressão dependente do resultado da primeira é feita por esse comando.

Por último, caso nenhuma das condições tenham sido verdadeiras, temos a estrutura *else* (no Português, “senão”), que executa um bloco de código caso nenhuma das outras condições não tenham sido realizadas.

A imagem mostra uma janela de terminal com um fundo escuro e uma barra de título cinza com três botões (vermelho, amarelo, verde). O código Python exibido é o seguinte:

```
1  codigo_ip = 'if669'
2
3  if codigo_ip == 'if669':
4      print(f'O código da disciplina Introdução a Programação é: {codigo_ip}')
5  else:
6      print('Esse não é o código de IP')
```

Figura 1: Exemplo de código com estruturas condicionais [1]

O resultado desse código será a impressão da frase "O código da disciplina Introdução a Programação é: if669", pois a primeira condição foi satisfeita.

O comando *print* é utilizado para dar saída, ou seja, fornecer um *output* para o usuário.

## 2.2 Laços de Repetição

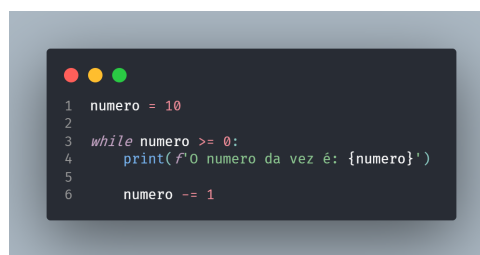
Existem algumas instruções dentro do nosso algoritmo que precisam ser executadas várias vezes seguidas no decorrer do código. Quando isso acontece, essa situação é chamada de *loop*. Para tal, no *Python*, têm-se dois comandos de repetição: o *for* e o *while*. O primeiro é geralmente usado quando se quer percorrer alguma coleção de dados, já o *while* é utilizado enquanto uma condição é verdadeira.

De maneira mais geral, é sempre explicado que o *for* é mais recomendável quando queremos iterar sobre uma sequência de dados. [2]

A imagem mostra uma janela de terminal com um fundo escuro e uma barra de título cinza com três botões (vermelho, amarelo, verde). O código Python exibido é o seguinte:

```
1  materias = ['md', 'ip', 'ic', 'avlc', 'calc']
2
3  for i in range(len(materias)):
4      print(materias[i])
```

Figura 2: Loop com *for*

A imagem mostra uma janela de terminal com um fundo escuro e uma barra de título cinza com três botões (vermelho, amarelo, verde). O código Python exibido é o seguinte:

```
1  numero = 10
2
3  while numero >= 0:
4      print(f'O numero da vez é: {numero}')
5
6      numero -= 1
```

Figura 3: Loop com *while*

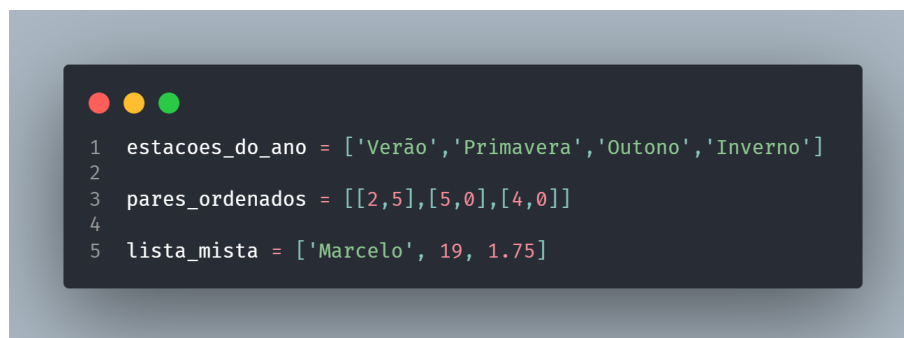
Como pode ser observado nas imagens, a 2 imprime todos os valores que estão dentro da lista "materias", incrementando o valor de 'i' a cada execução.

Já a 3 imprime o numero da vez enquanto o mesmo é maior ou igual a zero. Quando o numero for igual a -1, por exemplo, o *loop* não irá rodar mais.

O comando *len*, na figura 2 serve para mensurar o comprimento da lista, ou seja, para o valor i durante toda a extensão da lista materias, imprima o índice atual da lista.

## 2.3 Listas

Como visto no *loop "for"* do tópico anterior, listas são sequências de objetos, separados por vírgula e com começo e fim sinalizados através de colchetes. São utilizadas para armazenar diferentes valores, inclusive podem receber tipos de elementos variáveis.



```
1 estacoes_do_ano = ['Verão','Primavera','Outono','Inverno']
2
3 pares_ordenados = [[2,5],[5,0],[4,0]]
4
5 lista_mista = ['Marcelo', 19, 1.75]
```

Figura 4: Listas em *Python*

Na figura 4, pode-se observar três tipos de Listas:

1. A primeira lista só recebe valores em *strings* e armazena as estações do ano;
2. A lista 2 armazena coordenadas de um sistema X,Y. Pode-se notar, nesse exemplo, que existem listas dentro da lista maior;
3. No último exemplo, são armazenados diferentes tipos de variáveis na lista.

É importante dizer que listas são estruturas mutáveis, ou seja, podem ser acrescentadas de valores novos (isso é feito através do comando '*append*'), como também podem ser retirados valores dessa sequência (por meio do comando '*remove*').

## 2.4 Funções

Funções podem ser definidas como blocos de código que executam certa tarefa toda vez que são invocados no desenvolvimento do código. Isso significa que, ao criar um programa e realizar uma sequência de ações, esse mesmo processo pode ser realizado posteriormente no código ao defini-lo dentro de uma função. Basta, para executar aquela instrução novamente, invocar a função.

A sintaxe de uma função é feita por três partes: nome, parâmetros e corpo. O nome é importante para identificar a função, os parâmetros são os elementos necessários que aquela função precisa para ser executada, e o corpo é exatamente o que vai ser feito por essa estrutura.



Figura 5: Exemplo de Função

Na figura 5, podemos notar a definição de uma função com nome 'imprimir', que tem como parâmetros 'nome', 'idade' e 'trabalho'. Logo após, temos o corpo da função e, na linha 10, temos a chamada da função para executar suas instruções. Sempre que for necessário chamar essa função, basta apenas digitar seu nome e seus parâmetros.

## 2.5 Recursão

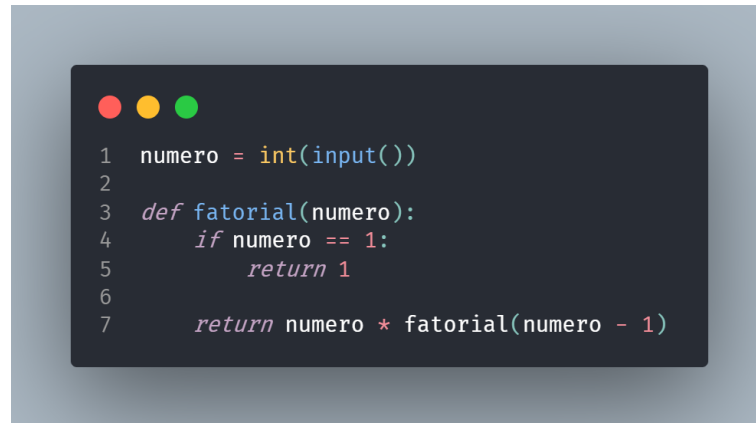
De acordo com o "*PensePython*",

Recursão é 'um método de resolução de problemas que envolve quebrar um problema em subproblemas menores e menores até chegar a um problema pequeno o suficiente para que ele possa ser resolvido trivialmente. Normalmente recursão envolve uma função que chama a si mesma. Embora possa não parecer muito, a recursão nos permite escrever soluções elegantes para problemas que, de outra forma, podem ser muito difíceis de programar.' [3]

A recursão tem 2 fatores principais:

1. Caso básico: quando a função vai parar
2. Casos recursivos: execução da função e chamada dela novamente, mudando os valores.

Como visto no tópico 2.4, uma função funciona como um método bastante importante para se criar nos códigos, podendo ser reutilizada durante todo o algoritmo. Por isso, criar uma recursão para essa função é muito mais intuitivo e enxuto em comparação com a criação de *loops*.

A screenshot of a Python code editor with a dark background and light-colored text. The code defines a recursive function 'fatorial' that takes a number as input and returns its factorial. The code is as follows:

```
1 numero = int(input())
2
3 def fatorial(numero):
4     if numero == 1:
5         return 1
6
7     return numero * fatorial(numero - 1)
```

Figura 6: Função recursiva em *Python*

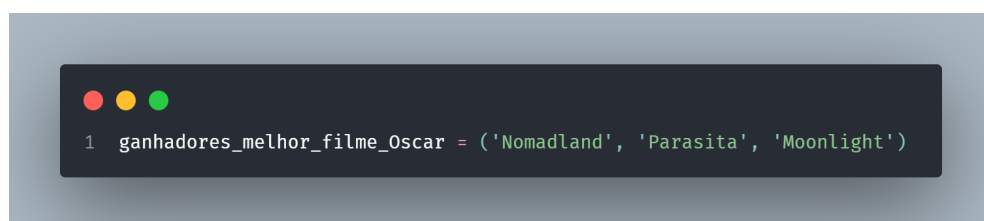
O caso da Figura 6 mostra a elaboração de uma função recursiva para cálculo do fatorial de um número inteiro.

A abreviação 'int' na variável 'número' significa que esse valor será do tipo inteiro, e o comando 'input' pede para o usuário digitar algum número. Após isso, tem-se a função 'fatorial', que calculará, a partir do *input*, o fatorial desse número,

## 2.6 Tuplas

Tuplas são estruturas de dados que também são capazes de armazenar dados, mas, diferentemente das Listas (abordadas no tópico 2.3, as tuplas são imutáveis, garantindo maior segurança no armazenamento de dados.

É possível, por exemplo, dizer que Getúlio Vargas não foi um dos presidentes do Brasil? Definitivamente não. Por isso, é mais viável armazenar esse tipo de informação dentro de uma estrutura que não é mutável e que garante que aqueles valores não serão modificados, e para isso temos a tupla.

A screenshot of a Python code editor with a dark background and light-colored text. The code creates a tuple named 'ganhadores\_melhor\_filme\_Oscar' containing the names of three movies: 'Nomadland', 'Parasita', and 'Moonlight'. The code is as follows:

```
1 ganhadores_melhor_filme_Oscar = ('Nomadland', 'Parasita', 'Moonlight')
```

Figura 7: Tupla em *Python*

Na figura acima, é inegável que tais filmes foram ganhadores da categoria 'Melhor Filme' em algumas edições do Oscar.

## 2.7 Dicionários

Se é desejado para o programa armazenar chaves e valores correspondentes para elas, existem os dicionários.

Essa estrutura de dados servem para armazenar uma chave e o(s) atributo(s) da mesma. Por exemplo, dentro do contexto da Figura 7, caso fosse necessário salvar todos os ganhadores de todas as categorias do Oscar 2019? Como isso poderia ser feito?.

O dicionário receberia, como chaves, todas as categorias, e os valores dessas chaves seriam os vencedores. A figura abaixo mostra, brevemente, como seria esse dicionário.



Figura 8: Exemplo de dicionário [4]

### 3 Relação com outras disciplinas e Relevância

'Introdução a Programação' é uma disciplina que desenvolve o discente para toda a trajetória acadêmica, haja visto que a lógica de resolução das listas de exercícios é exigida e ainda mais estimulada na cadeira 'Lógica para Computação', novos algoritmos e maior detalhamento sobre eles serão estudados em 'Algoritmos e Estrutura de Dados', assim como a maneira que a máquina lê o programa desenvolvido é visto na disciplina de 'Infraestrutura de Software'.

Além do mais, a definição de que *Python* é uma linguagem com tipagem dinâmica se relaciona com palestras ministradas para a disciplina de 'Introdução a Computação', a recursão, utilizada em funções recursivas, é abordada em 'Matemática Discreta para Computação' na elaboração de provas concretas para o campo da Matemática.

A disciplina 'Interface Usuário-Máquina' aborda, de acordo com o próprio site da cadeira [5], sobre “o escopo de conhecer as necessidades de um público e idealizar uma solução para a necessidade encontrada, conversando com pessoas que conhecem e/ou vivenciam o tema, para que a solução desenvolvida seja pertinente.” Portanto, os conhecimentos vistos em IP terão aplicação direta no progresso dessa disciplina.

O gerenciamento de dados também é abordado introdutoriamente em IP, tendo em vista que alguns comandos só são executados a partir da recepção de dados do(s) usuário(s), e a forma como esses dados são usados é explorada de maneira mais expansiva na disciplina *IF685*, 'Gerenciamento de Dados e Informação'.

Portanto, a conexão entre 'Introdução a Programação' e outras disciplinas é vasta, auxiliando o corpo discente na caminhada da graduação e também em projetos paralelos, assim como no mercado de trabalho.

## Referências

- [1] Todas as imagens referentes aos códigos do tópico 'Conteúdos' foram desenvolvidas pelo autor, com o uso do Visual Studio Code e a extensão 'CodeSnap' para gerar as imagens.
- [2] Erickson Lopes. Loops e estruturas de repetição. <https://pythonacademy.com.br/blog/estruturas-de-repeticao>, 2021. Acessado em 03/12/2021.
- [3] Panda IME USP. Recursão - Como pensar como um Cientista da Computação: Edição Interativa em *Python*. <https://panda.ime.usp.br/pensepy/static/pensepy/12-Recursao/recursionsimple-ptbr.html>, 2020. Acessado em 03/12/2021.
- [4] G1 Globo.com. Oscar 2019: veja os ganhadores. <https://g1.globo.com/pop-arte/cinema/oscar/2019/noticia/2019/02/24/oscar-2019-veja-os-ganhadores.ghhtml>, 2019. Acessado em 03/12/2021.
- [5] Centro de Informática UFPE. Interfaces usuários-máquina - cinwiki. [https://cin.ufpe.br/~pet/wiki/Interfaces\\_Usu%C3%A1rio-M%C3%A1quina](https://cin.ufpe.br/~pet/wiki/Interfaces_Usu%C3%A1rio-M%C3%A1quina), 2020. Acessado em 03/12/2021.