Estratégia Concursos

# Aula 00

*DataPrev (Analista de TI - Segurança Cibernética e Proteção de Dados) Passo Estratégico de Conhecimentos Específicos - 2024 (Pós-Edital)*

Autor:

13 de Setembro de 2024

# PYTHON

**Sumário**



# CONTEÚDO

Python. Tipos de dados. Variáveis. Operadores. Estruturas. Funções.

# ANÁLISE ESTATÍSTICA

Inicialmente, convém destacar o percentual de incidência do assunto, dentro da disciplina **Programação** em concursos/cargos similares. Quanto maior o percentual de cobrança de um dado assunto, maior sua importância.

Obs.: *um mesmo assunto pode ser classificado em mais de um tópico devido à multidisciplinaridade de conteúdo*.

| Assunto | Relevância na disciplina em concursos similares |
|---|---|
| Linguagens de programação | **38.2 %** |
| Python | **14.5 %** |
| JavaScript | **12.7 %** |
| Java | **3.6 %** |
| VB Script | **1.8 %** |
| C Sharp | **1.8 %** |
| R | **1.8 %** |
| Web | **5.5 %** |
| Linguagens de marcação | **5.5 %** |
| XML (Extensible Markup Language) | **3.6 %** |
| HTML (HyperText Markup Language) | **1.8 %** |
| CSS (Cascading Style Sheets) | **3.6 %** |
| Conceitos básicos de programação | **1.8 %** |
| Frameworks Java | **1.8 %** |
| Hibernate | **1.8 %** |

# GLOSSÁRIO DE TERMOS

*Faremos uma lista de termos que são relevantes ao entendimento do assunto desta aula. Caso tenha alguma dúvida durante a leitura, esta seção pode lhe ajudar a esclarecer.*

**Python**: Linguagem de programação de alto nível e fácil de aprender, utilizada em diversos tipos de projetos, desde análise de dados até desenvolvimento web.

**Tipo de dado**: Classificação dos valores que uma variável pode armazenar, como números, strings, booleanos, listas, dicionários, tuplas e conjuntos.

**Variável**: Nome que representa um valor armazenado na memória do computador.

**Número**: Tipo de dado que representa valores numéricos, como inteiros e decimais.

**String**: Tipo de dado que representa sequências de caracteres.

**Booleano**: Tipo de dado que representa valores verdadeiro ou falso.

**Lista**: Tipo de dado que representa uma coleção ordenada de valores.

**Dicionário**: Tipo de dado que representa uma coleção de valores indexados por chaves.

**Tupla**: Tipo de dado que representa uma sequência ordenada e imutável de valores.

**Conjunto**: Tipo de dado que representa uma coleção desordenada de valores únicos.

**Operador**: Símbolo ou palavra reservada que realiza uma operação em um ou mais valores.

**Operador aritmético**: Operador que realiza operações matemáticas, como adição, subtração, multiplicação e divisão.

**Operador de comparação**: Operador que compara dois valores e retorna um valor booleano indicando se são iguais, diferentes, maiores ou menores.

**Operador de atribuição**: Operador que atribui um valor a uma variável.

**Operador lógico**: Operador que combina valores booleanos e retorna um valor booleano, como and, or e not.

**Operador de string**: Operador que realiza operações em strings, como concatenação e busca de caracteres.

**Estrutura de seleção**: Estrutura que permite ao programa tomar decisões com base em condições, como if, else e if-elif-else.

**Estrutura de repetição**: Estrutura que permite ao programa repetir uma sequência de instruções enquanto uma condição é verdadeira, como for e while.

**Função**: Bloco de código que executa uma tarefa específica e pode ser reutilizado em diferentes partes do programa.

# ROTEIRO DE REVISÃO

*A ideia desta seção é apresentar um roteiro para que você realize uma revisão completa do assunto e, ao mesmo tempo, destacar aspectos do conteúdo que merecem atenção.*

## Python

Python é uma linguagem de programação de alto nível, interpretada, orientada a objetos e de propósito geral. Foi criada no final dos anos 1980 por Guido van Rossum e desde então se tornou uma das linguagens de programação mais populares do mundo.

Além de fácil de aprender, Python possui uma sintaxe clara e concisa, o que torna a codificação mais rápida e eficiente. Ela é usada em diversas áreas, incluindo desenvolvimento web, ciência de dados, inteligência artificial, automação de tarefas, jogos, entre outros.

Uma das principais características de Python é a sua comunidade ativa, que contribui para o desenvolvimento de bibliotecas e frameworks para a linguagem. Isso faz com que seja fácil encontrar recursos prontos para serem utilizados em projetos, reduzindo o tempo e o esforço de desenvolvimento.

## Características

Entre as características que facilitaram a adoção em larga escala de Python, podemos citar o fato dela ser de propósito geral, de alto nível e possuir uma excelente legibilidade. Veja cada uma a seguir:

**Propósito Geral**: Python é uma linguagem de propósito geral, o que significa que pode ser utilizada para desenvolver uma ampla variedade de aplicações. É possível criar desde pequenos scripts e programas simples até sistemas complexos e aplicações de grande escala.

**Linguagem de Alto Nível**: Python é considerada uma linguagem de alto nível porque oferece recursos que simplificam a codificação e reduzem a complexidade do código. Por exemplo, não é necessário gerenciar manualmente a alocação de memória ou lidar com detalhes de baixo nível. Além disso, Python possui uma sintaxe clara e concisa, o que torna o código mais legível e mais fácil de entender.

**Legibilidade**: A legibilidade é uma das características mais importantes de Python. A sintaxe clara e concisa da linguagem torna o código fácil de ler e entender, mesmo para aqueles que não estão familiarizados com a linguagem. Além disso, Python possui uma ampla variedade de recursos de documentação e comentários, o que torna mais fácil para os programadores documentarem seu código e torná-lo mais acessível para outros desenvolvedores.

## Aplicações

Python pode ser utilizada em diversas áreas. Aqui estão algumas das suas principais aplicações:

**Desenvolvimento Web**: Python é amplamente utilizada no desenvolvimento web, especialmente com o framework Django. Com o Django, é possível desenvolver rapidamente aplicações web robustas e escaláveis. Além disso, o Flask é outro framework popular em Python para desenvolvimento web.

**Ciência de Dados**: Com bibliotecas como o Pandas, Numpy e Matplotlib, é possível realizar análises de dados complexas, visualizar dados e criar modelos de aprendizado de máquina.

**Automação de Tarefas**: a linguagem é uma excelente escolha para automação de tarefas repetitivas e rotineiras. Com bibliotecas como o Selenium e o Beautiful Soup, é possível automatizar tarefas como web scraping, testes de software, automação de processos e muito mais.

**Inteligência Artificial**: Com bibliotecas como o TensorFlow, Keras e PyTorch, é possível criar modelos de aprendizado de máquina para reconhecimento de voz, imagem, processamento de linguagem natural, entre outros.

Bem, agora que já temos uma visão geral sobre as capacidades e características da linguagem, vamos aprofundar os seus detalhes, ok?

## Tipos de Dados e Variáveis

Os tipos de dados são os valores que podem ser armazenados em uma variável. Em Python, os tipos de dados mais comuns são:

**Números**: representam valores numéricos, como inteiros, float (números com ponto flutuante) e complexos.

```
idade = 23

altura = 1.75

salario = 5000.50

numero_complexo = 2 + 3j
```

**String**: representam texto, ou seja, uma sequência de caracteres. As strings são delimitadas por aspas simples ('') ou duplas ("").

```
nome = 'Maria'

sobrenome = "Silva"
```

**Booleanos**: representam valores lógicos, Verdadeiro (True) ou Falso (False).

```
estudando = True

trabalhando = False
```

**Listas**: representam uma coleção ordenada de valores que podem ser de diferentes tipos de dados. As listas são delimitadas por colchetes []

```
numeros = [1, 2, 3, 4, 5]

nomes = ['João', 'Maria', 'Pedro']
```

**Dicionários**: representam uma coleção de pares chave-valor. As chaves são únicas e são usadas para acessar os valores associados. Os dicionários são delimitados por chaves {}.

```
pessoa = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
```

**Tupla**: Uma tupla é uma coleção de valores ordenados e imutáveis que podem ser de diferentes tipos de dados. As tuplas são semelhantes às listas, mas ao contrário delas, as tuplas são imutáveis, ou seja, uma vez que uma tupla é criada, seus valores não podem ser alterados. As tuplas são delimitadas por parênteses ().

```
tupla1 = (1, 2, 3)

tupla2 = ('João', 30, 1.75)
```

**Conjunto**: Um conjunto é uma coleção não ordenada e não indexada de elementos únicos. Os conjuntos são usados quando é importante que os elementos sejam únicos, sem ordem específica. Os conjuntos são delimitados por chaves {}.

```
conjunto1 = {1, 2, 3}

conjunto2 = {'a', 'b', 'c'}
```

**Resumo dos tipos de dados (com exemplos):**

| Tipo de Dado | Descrição | Exemplo |
|---|---|---|
| Inteiro | Números inteiros, positivos ou negativos | 10, -20, 0 |
| Float | Números com ponto flutuante | 3.14, -2.5, 1.0 |
| Complexo | Números complexos no formato real + imaginário*j | 2+3j, -1+4j, 1j |
| String | Sequência de caracteres | 'Olá, mundo!', "Python" |
| Booleano | Valores lógicos Verdadeiro (True) ou Falso (False) | True, False |
| Lista | Coleção ordenada de valores de diferentes tipos | [1, 2, 3], ['maçã', 'banana'] |
| Tupla | Coleção ordenada imutável de valores de diferentes tipos | (1, 2, 3), ('João', 30) |
| Dicionário | Coleção de pares chave-valor | {'nome': 'João', 'idade': 30} |
| Conjunto | Coleção não ordenada de elementos únicos | {1, 2, 3}, {'a', 'b', 'c'} |

### Variáveis

Uma variável é um local de armazenamento na memória do computador que contém um valor. Ela é utilizada para representar um valor específico que pode ser alterado ao longo do programa. Cada variável tem um nome único que é usado para acessar e manipular o valor que ela contém.

Veja o exemplo:

```
nome = 'João'

idade = 30

altura = 1.75
```

Nesse exemplo, criamos três variáveis chamadas nome, idade e altura e atribuímos a elas os valores 'João', 30 e 1.75, respectivamente. Essas variáveis podem ser usadas posteriormente no código para realizar cálculos, exibir informações na tela, etc.

As variáveis em Python são dinamicamente tipadas, o que significa que não é necessário especificar o tipo de dados ao declará-las. O tipo de dados da variável é determinado automaticamente pelo valor atribuído a ela.

## Operadores

Operadores são símbolos que indicam uma operação a ser realizada entre dois ou mais valores em uma expressão ou instrução. Em Python, há diversos tipos de operadores, que podem ser utilizados para realizar operações aritméticas, lógicas, de comparação, de atribuição, entre outras.

### Operadores aritméticos

São utilizados para realizar operações matemáticas básicas, como adição (+), subtração (-), multiplicação (*), divisão (/) e outros.

| Operador | Significado | Exemplo |
|---|---|---|
| + | Adição | 3 + 5 = 8 |
| - | Subtração | 7 - 2 = 5 |
| * | Multiplicação | 2 * 4 = 8 |
| / | Divisão | 10 / 2 = 5 |
| % | Módulo | 11 % 3 = 2 |
| ** | Potência | 2 ** 3 = 8 |

## Operadores de comparação

São utilizados para comparar dois valores e retornar um valor booleano Verdadeiro (True) ou Falso (False).

| Operador | Significado | Exemplo |
|---|---|---|
| == | Igualdade | 3 == 5 → False |
| != | Diferença | 7 != 2 → True |
| > | Maior que | 10 > 2 → True |
| < | Menor que | 11 < 3 → False |
| >= | Maior ou igual a | 5 >= 5 → True |
| <= | Menor ou igual a | 8 <= 2 → False |

## Operadores de atribuição

São utilizados para atribuir um valor a uma variável. O operador de atribuição básico é o sinal de igual (=), mas há também outros operadores de atribuição que podem ser combinados com operações aritméticas, como soma (+=), subtração (-=), multiplicação (*=) e divisão (/=).

| Operador | Significado | Exemplo |
|---|---|---|
| = | Atribuição | x = 10 |
| += | Adição | x += 5 (equivalente a x = x + 5) |
| -= | Subtração | x -= 2 (equivalente a x = x - 2) |
| *= | Multiplicação | x *= 3 (equivalente a x = x * 3) |
| /= | Divisão | x /= 2 (equivalente a x = x / 2) |
| %= | Módulo | x %= 3 (equivalente a x = x % 3) |
| //= | Divisão inteira | x //= 2 (equivalente a x = x // 2) |
| **= | Potência | x **= 2 (equivalente a x = x ** 2) |

## Operadores lógicos

Operadores lógicos são usados para combinar expressões booleanas e obter um resultado booleano. Em Python, existem três operadores lógicos: and, or e not.

O operador **and** retorna True se ambas as expressões forem verdadeiras. Caso contrário, retorna False.

O operador **or** retorna True se pelo menos uma das expressões for verdadeira. Se ambas as expressões forem falsas, retorna False.

O operador **not** inverte o valor da expressão, ou seja, se a expressão for True, retorna False, e se for False, retorna True.

Aqui está uma tabela com exemplos de como esses operadores podem ser utilizados:

| Expressão 1 | Expressão 2 | and | or | not (Expressão 1) |
|---|---|---|---|---|
| True | True | True | True | False |
| True | False | False | True | False |
| False | True | False | True | True |

| | | | | |
|---|---|---|---|---|
| **False** | **False** | **False** | **False** | **True** |

Na primeira e segunda colunas, temos as expressões booleanas que serão combinadas pelos operadores lógicos. Na terceira, quarta e quinta colunas, temos o resultado da combinação das expressões com os operadores lógicos and, or e not, respectivamente.

Por exemplo, na primeira linha da tabela, temos a combinação de duas expressões True com o operador and. Como ambas as expressões são verdadeiras, o resultado também é verdadeiro. Na segunda linha, temos a combinação de uma expressão True com uma expressão False com o operador or. Como pelo menos uma das expressões é verdadeira, o resultado também é verdadeiro. Na quinta linha, temos o operador not sendo aplicado a uma expressão False. Como a expressão original é falsa, o resultado do operador not é verdadeiro.

## Operadores de Strings

Em Python, os operadores sobre strings são usados para realizar várias operações, como comparação, concatenação e busca de caracteres. Aqui estão alguns exemplos desses operadores:

**Operador de comparação**: ==

O operador de comparação == é usado para comparar duas strings e verificar se elas são iguais. Ele retorna True se as strings forem iguais e False caso contrário. Por exemplo:

```
string1 = "hello"
string2 = "hello"
string3 = "world"

print(string1 == string2)  # True
print(string1 == string3)  # False
```

**Operador de concatenação**: +

O operador de concatenação + é usado para unir duas strings em uma única string. Por exemplo:

```
string1 = "hello"

string2 = "world"


string3 = string1 + " " + string2

print(string3)  # "hello world"
```

**Operador de busca de caracteres**: in

O operador in é usado para verificar se um determinado caractere ou substring está presente em uma string. Ele retorna True se o caractere ou substring estiver presente e False caso contrário. Por exemplo:

```
string1 = "hello"

string2 = "world"


print("h" in string1)  # True

print("o" in string2)  # True

print("x" in string1)  # False
```

Existem também outros operadores que podem ser usados para manipular strings em Python, como o operador de multiplicação *, que é usado para criar uma nova string a partir

da repetição de uma string existente, e o operador de fatiamento [], que é usado para obter um subconjunto de uma string com base em sua posição.

## Estruturas de Seleção

As estruturas condicionais **if, else e if-elif-else** são usadas em Python para executar diferentes blocos de código com base em uma condição.

**if:**

O if é uma estrutura condicional simples que verifica uma condição e executa um bloco de código somente se a condição for verdadeira. O formato básico do if é:

```
if condição:
    bloco de código
```

Por exemplo:

```
idade = 20
if idade >= 18:
    print("Você é maior de idade")
```

**else:**

O else é usado juntamente com o if para executar um bloco de código diferente quando a condição do if não é verdadeira. O formato básico do else é:

```
if condição:
    bloco de código se a condição for verdadeira
else:
    bloco de código se a condição for falsa
```

Por exemplo:

```
idade = 16
if idade >= 18:
    print("Você é maior de idade")
else:
    print("Você é menor de idade")
```

**if-elif-else:**

O if-elif-else é uma estrutura condicional mais complexa que permite testar várias condições diferentes e executar um bloco de código correspondente a cada condição. O formato básico do if-elif-else é:

```
if condição1:
    bloco de código se condição1 for verdadeira
elif condição2:
```

```
    bloco de código se condição2 for verdadeira
elif condição3:
    bloco de código se condição3 for verdadeira
else:
    bloco de código se todas as condições forem falsas
```

Por exemplo:

```
nota = 7
if nota >= 9:
    print("Parabéns! Sua nota foi A")
elif nota >= 7:
    print("Sua nota foi B")
elif nota >= 5:
    print("Sua nota foi C")
else:
    print("Infelizmente você foi reprovado")
```

As estruturas condicionais if, else e if-elif-else são muito importantes para o desenvolvimento de programas em Python, pois permitem que o programa execute diferentes ações com base em condições específica.

## Estruturas de Repetição

As estruturas de repetição (também chamadas de laços ou loops) permitem que um trecho de código seja executado várias vezes. Em Python, existem duas estruturas de repetição: o for e o while.

A estrutura `for` é usada para iterar sobre uma sequência (como uma lista, tupla, string etc.) e executar um bloco de código para cada item da sequência. A sintaxe básica é:

```
for item in sequência:
    # código a ser executado para cada item
```

Já a estrutura `while` é usada para repetir um bloco de código enquanto uma determinada condição for verdadeira. A sintaxe básica é:

```
while condição:
    # código a ser repetido enquanto a condição for verdadeira
```

Segue abaixo uma tabela com exemplos de uso das estruturas de repetição em Python:

| Estrutura | Exemplo | Descrição |
|---|---|---|
| for | `for i in range(5): print(i)` | Itera sobre a sequência `range(5)` e imprime cada valor |
| for | `for letra in 'python': print(letra)` | Itera sobre a string `'python'` e imprime cada letra |
| while | `x = 0; while x < 5: print(x); x += 1` | Imprime os valores de `x` de 0 a 4 |

| | | |
|---|---|---|
| while | `while True: senha = input('Digite a senha: '); if senha == '123': break` | Repete a solicitação de uma senha até que o valor digitado seja igual a `'123'` |
| for | `for x, y in zip(lista1, lista2): print(x, y)` | Itera simultaneamente sobre duas listas (`lista1` e `lista2`) e imprime os valores correspondentes |

Esses são apenas alguns exemplos de como as estruturas de repetição podem ser utilizadas em Python. É importante entender bem o funcionamento de cada uma delas para poder escolher a melhor opção para cada caso específico.

## Funções

Em Python, funções são blocos de código que podem ser reutilizados várias vezes no programa. Elas são uma parte fundamental da programação modular, que divide o programa em partes menores e mais gerenciáveis.

Uma função pode receber um ou mais argumentos de entrada e pode retornar um ou mais valores de saída. Elas são declaradas usando a palavra-chave def seguida pelo nome da função, parênteses e dois pontos. O bloco de código da função é indentado abaixo da declaração da função.

Por exemplo, vamos criar uma função simples que recebe dois números e retorna a sua soma:

```
def soma(a, b):
    resultado = a + b
    return resultado
```

Agora podemos chamar a função soma com diferentes argumentos e obter o resultado da soma:

```
print(soma(2, 3)) # imprime 5

print(soma(10, 20)) # imprime 30
```

As funções são uma maneira poderosa de simplificar o código e torná-lo mais fácil de entender e manter. Elas permitem que blocos de código sejam reutilizados várias vezes, reduzindo a quantidade de código redundante no programa. Além disso, as funções permitem que o programador abstraia a complexidade do código e trabalhe em níveis mais altos de abstração, o que pode tornar o código mais fácil de entender e depurar.

Usando funções, é possível dividir um programa em partes menores, cada uma com sua própria responsabilidade e funcionalidade, o que torna o programa mais fácil de entender, manter e atualizar.

### Funções de usuário x Funções pré-definidas

Funções pré-definidas (também chamadas de funções built-in) são funções que já estão disponíveis em Python, sem a necessidade de defini-las previamente. Elas são funções que já foram criadas pelos desenvolvedores Python e estão disponíveis para serem utilizadas em qualquer programa. Alguns exemplos de funções pré-definidas incluem print(), len(), str(), int(), float(), entre outras.

Por outro lado, as funções de usuário são funções criadas pelo próprio programador para realizar tarefas específicas em seu programa. Essas funções devem ser definidas pelo programador antes de serem usadas.

Veja alguns exemplos de funções pré-definidas em Python:

| Função | Descrição | Exemplo |
|---|---|---|
| `print()` | Imprime uma mensagem na tela | `print("Olá, mundo!")` |
| `len()` | Retorna o tamanho de um objeto | `len("Python")` |
| `range()` | Gera uma sequência de números | `range(0, 10)` |
| `type()` | Retorna o tipo de um objeto | `type(10)` |
| `input()` | Lê uma entrada do usuário | `nome = input("Qual é o seu nome? ")` |
| `str()` | Converte um objeto para uma string | `str(10)` |

| | | |
|---|---|---|
| `int()` | Converte um objeto para um inteiro | `int("10")` |
| `float()` | Converte um objeto para um número de ponto flutuante | `float("3.14")` |
| `max()` | Retorna o maior valor em um objeto | `max([10, 20, 30])` |
| `min()` | Retorna o menor valor em um objeto | `min([10, 20, 30])` |
| `sum()` | Retorna a soma de todos os valores em um objeto | `sum([1, 2, 3, 4])` |
| `sorted()` | Retorna uma lista ordenada de valores em um objeto | `sorted([3, 1, 4, 1, 5, 9, 2, 6, 5])` |

# APOSTA ESTRATÉGICA

*A ideia desta seção é apresentar os pontos do conteúdo que mais possuem chances de serem cobrados em prova, considerando o histórico de questões da banca em provas de nível semelhante à nossa, bem como as inovações no conteúdo, na legislação e nos entendimentos doutrinários e jurisprudenciais[1].*

Python oferece uma variedade de tipos de dados incorporados que facilitam a manipulação de informações em diferentes contextos. Os tipos de dados básicos incluem:

Inteiros (int): São números sem parte decimal, utilizados para contar ou indexar coleções. Python suporta inteiros de tamanho arbitrário.

Números de ponto flutuante (float): Representam números reais e são usados para cálculos que requerem frações.

Booleanos (bool): Possuem apenas dois valores, True ou False, e são frequentemente empregados em condições e loops.

Strings (str): Sequências de caracteres usadas para armazenar texto.

Listas (list): Coleções ordenadas e mutáveis de itens que podem ser de tipos diferentes.

Tuplas (tuple): Coleções ordenadas e imutáveis. Semelhantes às listas, mas não podem ser alteradas após sua criação.

[1] Vale deixar claro que nem sempre será possível realizar uma aposta estratégica para um determinado assunto, considerando que às vezes não é viável identificar os pontos mais prováveis de serem cobrados a partir de critérios objetivos ou minimamente razoáveis.

Dicionários (dict): Estruturas de dados que armazenam pares chave-valor, permitindo a rápida recuperação de dados com base na chave.

Conjuntos (set): Coleções não ordenadas de elementos únicos que são úteis para testar pertencimento, remover duplicatas e realizar operações matemáticas como união e interseção.

# QUESTÕES ESTRATÉGICAS

*Nesta seção, apresentamos e comentamos uma amostra de questões objetivas selecionadas estrategicamente: são questões com nível de dificuldade semelhante ao que você deve esperar para a sua prova e que, em conjunto, abordam os principais pontos do assunto*

*A ideia, aqui, não é que você fixe o conteúdo por meio de uma bateria extensa de questões, mas que você faça uma boa revisão global do assunto a partir de, relativamente, poucas questões.*

1. **(FGV / FUNSAÚDE - 2021)** Observe o código Python v2.7.

def F (a, b): while a != b: if a > b: a = a - b elif b > a: b -= a return a

Assinale o valor retornado para F (48,36).

a) 1
b) 12
c) 24
d) 36
e) 48

**Comentários**:

Vamos lá! A função F(a,b), foi invocada com a = 48 e b = 36. Antes de entrarmos nas iterações, é bom apenas deixar claro que b -= a é o mesmo que b = b-a. Vamos agora para as iterações:

48 != 36?
Sim, logo vamos para a próxima linha:
48>36? Sim, logo a = 48 – 36 = 12.
12 != 36?
Sim, logo vamos para a próxima linha:
12 > 36? Não, então 36 > 12? Sim, logo b = 36 – 12 = 24.
12 != 24?
Sim, logo vamos para a próxima linha:
12 > 24? Não, então 24 > 12? Sim, logo b = 24 – 12 = 12.

12 != 12?
Não, 12 = 12. Logo, retornamos a, que é = 12.

**Gabarito**: B

2. **(FGV – CM/Caruaru – 2015)** Analise o código Python a seguir.

```
L1=[10,20,30]
L2=[40,50]
L1.append(L2)
print L1
```

Assinale a opção que descreve corretamente o que acontece quando esse programa é executado no Python 2.7:

a) Produz uma mensagem de erro, porque tenta executar uma operação inválida.
b) Exibe "[10, 20, 30, [40, 50]]".
c) Exibe "[10, 20, 30, 40, 50]".
d) Exibe "[10, 20, 30], [40, 50]".
e) Exibe "[]".

**Comentários**:

O comando append inclui o valor da variável L2 na posição final do vetor L1. Como o conteúdo da variável L2 é uma lista de tamanho 2 [40,50], ele que será incluído na 4ª posição de L1. Professor, porque a resposta não é letra C? Muito bem observado, padawan! O método append inclui a lista L2 como se fosse só um elemento, isto é, a lista final tem 4 elementos ao invés de 5, pois L2 é tratado como se fosse uma coisa só!

**Gabarito**: B

3. **(FGV / Receita Federal - 2023)** Analise o código Python a seguir.

```
class enigma:

def __iter__(self):

self.x = 128

return self

def __next__(self):

y = self.x

z = lambda a : a - int(a/2)
```

self.x = z(self.x)

return y

coisa = enigma()

xpto = iter(coisa)

for k in range(5):

print(next(xpto))

Assinale o quarto número exibido na execução desse código.

a) 4

b) 8

c) 16

d) 32

e) 64

**Comentários:**

A classe "enigma" é definida com dois métodos especiais em Python: "iter" e "next". Esses métodos são usados para tornar a classe um iterador, que é uma estrutura de dados que permite que você percorra os elementos de uma sequência.

No método "iter", o valor inicial de "self.x" é definido como 128 e a função retorna a instância do objeto iterador, ou seja, "self". No método "next", a variável "y" recebe o valor atual de "self.x", que é 128 na primeira iteração. A variável "z" é definida como uma função lambda que divide o valor de entrada por 2 e subtrai a parte inteira do resultado. Então, "self.x" é atualizado para o resultado da função "z" com "self.x" como entrada. Vejamos as iterações:

**1ª iteração:**

self.x = 128

z(self.x) = 64

y = self.x = 128

**Valor impresso: 128**

**2ª iteração:**

self.x = 64

z(self.x) = 32

y = self.x = 64

**Valor impresso: 64**

**3ª iteração:**

self.x = 32

z(self.x) = 16

y = self.x = 32

**Valor impresso: 32**

**4ª iteração:**

self.x = 16

z(self.x) = 8

y = self.x = 16

**Valor impresso: 16**

**Gabarito:** Letra C

4. **(FGV / Receita Federal - 2023)** Analise o código Python a seguir.

```
L=[]
for x in range(10,1,-2):
  L.append(x)
print (L[2:4])
```

Assinale a opção que indica os valores exibidos na execução desse código.

a) [4, 6]

b) [6, 2]

c) [6, 4]

d) [8, 4]

e) [8, 6, 4, 2]

**Comentários:**

O código em Python começa criando uma lista vazia chamada "L" através da seguinte linha de código:

L=[]

Em seguida, o código utiliza um loop "for" para preencher a lista com valores que começam em 10 e decrementam de 2 em 2 até chegar a 2. A linha de código responsável por essa tarefa é:

```
for x in range(10, 1, -2):
    L.append(x)
```

Neste loop, a função range(10, 1, -2) gera uma sequência de números que começa em 10 e termina em 2 (excluindo o 2), com um passo de -2. Então, a variável "x" assume valores dessa sequência em cada iteração do loop, e o comando L.append(x) adiciona o valor atual de "x" na lista "L". Dessa forma, após a execução desse loop, a lista "L" ficará com os valores [10, 8, 6, 4, 2]. Por fim, o código exibe na tela os elementos da lista que estão nas posições 2 e 3 utilizando o seguinte comando:

```
print(L[2:4])
```

A expressão "L[2:4]" é conhecida como slicing em Python e retorna um subconjunto da lista "L" que começa na posição 2 (incluindo) e termina na posição 4 (excluindo), ou seja, contém os elementos de índice 2 e 3 da lista original. Assim, a saída do programa é a lista [6, 4].

**Gabarito:** Letra C

5. **(FGV / Receita Federal - 2023)** Analise o código Python a seguir.

```
def xxx(a, b):
  while b != 0:
        a, b = b, a % b
  return a
print (xxx(90,15))
```

Assinale o resultado exibido na execução desse código.

a) 1

b) 3

c) 6

d) 15

e) 75

**Comentários:**

O código Python apresentado define uma função chamada "xxx" que recebe dois parâmetros "a" e "b". Dentro da função, é executado um loop "while" que continua enquanto "b" for diferente de zero. Em cada iteração do loop, as variáveis "a" e "b" são atualizadas para "b" e "a % b", respectivamente, onde "% " representa o operador módulo, ou seja, o resto da divisão de "a" por "b".

O loop continua até que "b" seja igual a zero, e o último valor atribuído à variável "a" é retornado como resultado da função. No código em questão, a função "xxx" é chamada com os valores 90 e 15 como argumentos, ou seja, a = 90 e b = 15. Nesse caso, o loop da função executará 2 iterações antes de b chegar a zero, como podemos ver abaixo:

**Iteração 1:** a = 90, b = 15

**Iteração 2:** a = 15, b = 90 % 15 = 0

Na última iteração, "b" é igual a zero, então a função retorna o valor atual de "a", que é 15. Sendo assim, o resultado exibido na execução desse código é 15.

**Gabarito:** Letra D

# QUESTIONÁRIO DE REVISÃO E APERFEIÇOAMENTO

*A ideia do questionário é elevar o nível da sua compreensão no assunto e, ao mesmo tempo, proporcionar uma outra forma de revisão de pontos importantes do conteúdo, a partir de perguntas que exigem respostas subjetivas.*

*São questões um pouco mais desafiadoras, porque a redação de seu enunciado não ajuda na sua resolução, como ocorre nas clássicas questões objetivas.*

*O objetivo é que você realize uma auto explicação mental de alguns pontos do conteúdo, para consolidar melhor o que aprendeu ;)*

*Além disso, as questões objetivas, em regra, abordam pontos isolados de um dado assunto. Assim, ao resolver várias questões objetivas, o candidato acaba memorizando pontos isolados do conteúdo, mas muitas vezes acaba não entendendo como esses pontos se conectam.*

*Assim, no questionário, buscaremos trazer também situações que ajudem você a conectar melhor os diversos pontos do conteúdo, na medida do possível.*

*É importante frisar que não estamos adentrando em um nível de profundidade maior que o exigido na sua prova, mas apenas permitindo que você compreenda melhor o assunto de modo a facilitar a resolução de questões objetivas típicas de concursos, ok?*

*Nosso compromisso é proporcionar a você uma revisão de alto nível!*

*Vamos ao nosso questionário:*

## Perguntas

**1.** O que é Python?

**2.** O que são tipos de dados em Python?

**3.** O que é uma variável em Python?

**4.** O que são operadores em Python?

**5.** Qual a diferença entre uma tupla e um conjunto em Python?

**6.** O que é uma estrutura condicional em Python?

**7.** Por que devemos usar funções em Python?

**8.** Qual a diferença entre uma função pré-definida e uma função de usuário em Python?

**9.** O que é uma estrutura de repetição em Python?

**10.** Quais são os operadores lógicos em Python?

**11.** O que são operadores de string em Python?

**12.** O que são funções built-in em Python?

**13.** Quais são os principais tipos de dados em Python e qual a diferença entre eles?

**14.** Quais são os operadores de comparação em Python e quais os resultados possíveis?

**15.** Qual a diferença entre uma estrutura de seleção com "if" e uma estrutura de seleção com "if-elif-else" em Python?

**16.** Por que devemos usar funções em Python?

**17.** O que são estruturas de repetição em Python e por que elas são importantes?

**18.** Explique a diferença entre o "for" e o "while" em Python.

**19.** O que é a precedência de operadores em Python?

**20.** O que é uma lista em Python e como ela é representada?

**21.** O que é um conjunto em Python e quais são suas características?

**22.** O que é um dicionário em Python e como ele é estruturado?

## Perguntas e Respostas

**1.** O que é Python?

Resposta: Python é uma linguagem de programação de alto nível, interpretada e de propósito geral.

**2.** O que são tipos de dados em Python?

Resposta: Os tipos de dados em Python são utilizados para representar os diferentes valores que uma variável pode armazenar.

**3.** O que é uma variável em Python?

Resposta: Uma variável em Python é um espaço de memória reservado para armazenar um valor.

**4.** O que são operadores em Python?

Resposta: Operadores em Python são símbolos ou palavras reservadas que são utilizados para executar operações em variáveis e valores.

**5.** Qual a diferença entre uma tupla e um conjunto em Python?

Resposta: A principal diferença é que uma tupla é imutável, enquanto um conjunto é mutável.

**6.** O que é uma estrutura condicional em Python?

Resposta: Uma estrutura condicional em Python é uma estrutura de controle que permite que o código execute diferentes ações com base em uma condição.

**7.** Por que devemos usar funções em Python?

Resposta: Funções em Python nos permitem reutilizar o código e torná-lo mais legível e organizado.

**8.** Qual a diferença entre uma função pré-definida e uma função de usuário em Python?

Resposta: As funções pré-definidas são fornecidas pela linguagem Python, enquanto as funções de usuário são criadas pelo programador.

**9.** O que é uma estrutura de repetição em Python?

Resposta: Uma estrutura de repetição em Python é uma estrutura de controle que permite que o código execute repetidamente até que uma condição seja atendida.

**10.** Quais são os operadores lógicos em Python?

Resposta: Os operadores lógicos em Python são "and", "or" e "not".

**11.** O que são operadores de string em Python?

Resposta: Os operadores de string em Python permitem que as strings sejam concatenadas, comparadas e acessadas.

**12.** O que são funções built-in em Python?

Resposta: Funções built-in em Python são funções pré-definidas que já estão disponíveis na linguagem Python.

**13.** Quais são os principais tipos de dados em Python e qual a diferença entre eles?

Resposta: Os principais tipos de dados em Python são inteiros (int), números de ponto flutuante (float), strings (str), booleanos (bool), listas (list), dicionários (dict), tuplas (tuple) e conjuntos (set). A diferença entre eles está na forma como os valores são armazenados e manipulados. Por exemplo, as listas são mutáveis e podem armazenar vários valores, enquanto as tuplas são imutáveis e contêm uma sequência fixa de valores.

**14.** Quais são os operadores de comparação em Python e quais os resultados possíveis?

Resposta: Os operadores de comparação em Python são == (igual), != (diferente), > (maior que), < (menor que), >= (maior ou igual a) e <= (menor ou igual a). O resultado de uma comparação é um valor booleano, ou seja, True (verdadeiro) se a comparação for verdadeira e False (falso) caso contrário.

**15.** Qual a diferença entre uma estrutura de seleção com "if" e uma estrutura de seleção com "if-elif-else" em Python?

Resposta: A estrutura "if" é utilizada para tomar decisões com base em uma única condição. Se a condição for verdadeira, o bloco de código associado ao "if" é executado. Já a estrutura "if-elif-else" permite avaliar várias condições em sequência. Se uma condição for verdadeira, o bloco de código associado a essa condição é executado e as condições restantes não são verificadas.

**16.** Por que devemos usar funções em Python?

Resposta: Funções em Python nos permitem organizar o código de forma modular e reutilizável. Elas dividem tarefas complexas em partes menores e mais gerenciáveis, facilitando a leitura, manutenção e depuração do código. Além disso, as funções permitem que blocos de código sejam executados várias vezes em diferentes partes do programa, evitando repetição de código.

**17.** O que são estruturas de repetição em Python e por que elas são importantes?

Resposta: As estruturas de repetição em Python permitem que um bloco de código seja executado repetidamente até que uma determinada condição seja atendida. Elas são importantes porque automatizam tarefas repetitivas, economizando tempo e esforço do programador. As estruturas de repetição mais comuns em Python são o "for" e o "while".

**18.** Explique a diferença entre o "for" e o "while" em Python.

Resposta: O "for" em Python é usado para percorrer uma sequência de elementos, como uma lista, tupla ou string. Ele executa um bloco de código para cada elemento na sequência. Já o "while" é usado quando queremos repetir um bloco de código enquanto uma determinada condição for verdadeira. A condição é verificada antes de cada iteração.

**19.** O que é a precedência de operadores em Python?

Resposta: A precedência de operadores em Python determina a ordem em que as operações são executadas em uma expressão. Alguns operadores têm prioridade sobre outros. Por exemplo, o operador de multiplicação (*) tem uma precedência maior do que o operador de adição (+), portanto, ele será executado primeiro.

**20.** O que é uma lista em Python e como ela é representada?

Resposta: Uma lista em Python é uma estrutura de dados que permite armazenar uma coleção de valores. Ela pode conter elementos de diferentes tipos, como números, strings e até mesmo outras listas. As listas são representadas por valores entre colchetes [], separados por vírgulas.

**21.** O que é um conjunto em Python e quais são suas características?

Resposta: Um conjunto em Python é uma coleção desordenada de elementos únicos. Ele não permite duplicatas, o que o torna útil para operações de união, interseção e diferença entre conjuntos. Os conjuntos são representados por valores entre chaves {}, separados por vírgulas.

**22.** O que é um dicionário em Python e como ele é estruturado?

Resposta: Um dicionário em Python é uma estrutura de dados que mapeia chaves a valores. Cada valor é associado a uma chave única, permitindo um acesso eficiente aos dados. Os dicionários são representados por pares de chave-valor, separados por vírgulas e delimitados por chaves {}. Por exemplo: {"nome": "João", "idade": 30}.

## LISTA DE QUESTÕES ESTRATÉGICAS

1. **(FGV / FUNSAÚDE - 2021)** Observe o código Python v2.7.

   def F (a, b): while a != b: if a > b: a = a - b elif b > a: b -= a return a

   Assinale o valor retornado para F (48,36).

   a) 1
   b) 12
   c) 24
   d) 36
   e) 48

2. **(FGV – CM/Caruaru – 2015)** Analise o código Python a seguir.

```
L1=[10,20,30]
L2=[40,50]
L1.append(L2)
print L1
```

Assinale a opção que descreve corretamente o que acontece quando esse programa é executado no Python 2.7:

a) Produz uma mensagem de erro, porque tenta executar uma operação inválida.
b) Exibe “[10, 20, 30, [40, 50]]”.
c) Exibe “[10, 20, 30, 40, 50]”.
d) Exibe “[10, 20, 30], [40, 50]”.
e) Exibe “[]”.

3. **(FGV / Receita Federal - 2023)** Analise o código Python a seguir.

```
class enigma:

def __iter__(self):

self.x = 128

return self

def __next__(self):

y = self.x

z = lambda a : a - int(a/2)

self.x = z(self.x)

return y

coisa = enigma()

xpto = iter(coisa)

for k in range(5):

print(next(xpto))
```

Assinale o quarto número exibido na execução desse código.

a) 4

b) 8

c) 16

d) 32

e) 64

4. **(FGV / Receita Federal - 2023)** Analise o código Python a seguir.

```
L=[]
for x in range(10,1,-2):
  L.append(x)
print (L[2:4])
```

Assinale a opção que indica os valores exibidos na execução desse código.

a) [4, 6]

b) [6, 2]

c) [6, 4]

d) [8, 4]

e) [8, 6, 4, 2]

5. **(FGV / Receita Federal - 2023)** Analise o código Python a seguir.

```
def xxx(a, b):
  while b != 0:
        a, b = b, a % b
  return a
print (xxx(90,15))
```

Assinale o resultado exibido na execução desse código.

a) 1

b) 3

c) 6

d) 15

e) 75

Gabaritos

1. B

2. B

3. C

4. C

5. D

## Questões Adicionais

*As questões apresentadas a seguir integram o Banco de Questões do Passo Estratégico. Recomenda-se utilizá-las como um recurso complementar para a prática e consolidação dos conhecimentos adquiridos no material teórico, de acordo com o estilo adotado pela banca organizadora.*

*Bom estudo!*

**1.** Qual é a saída do seguinte código em Python?

```
x = [1, 2, 3]
x.append([4, 5])
print(len(x))
```

A) 3
B) 4
C) 5
D) 6
E) 7

**2.** Em Python, qual é a função principal do operador '=='?

A) Atribuir valores a variáveis.
B) Concatenar duas strings.
C) Verificar se uma variável é maior que outra.
D) Comparar duas variáveis para verificar se são iguais.
E) Dividir dois números inteiros.

**3.** No Python, qual é o resultado da seguinte expressão: 18 % 7?

A) 0
B) 1
C) 2
D) 3
E) 4

**4.** Quando se utiliza um dicionário em Python, é possível obter uma lista de todas as chaves presentes no dicionário. Qual das opções a seguir demonstra corretamente como fazer isso utilizando o método apropriado?

A) chaves = dict.chaves()
B) chaves = dict.values()
C) chaves = list(dict)
D) chaves = dict.items()
E) chaves = dict.keys()

**5.** Considere o seguinte trecho de código em Python: 'minha_lista = [1, 2, 3, 4, 5]; nova_lista = [x*2 for x in minha_lista]'. Qual será o conteúdo da variável 'nova_lista' após a execução deste código?

A) [1, 2, 3, 4, 5]
B) [2, 4, 6, 8, 10]
C) [3, 6, 9, 12, 15]
D) [4, 8, 12, 16, 20]
E) [5, 10, 15, 20, 25]

**6.** No contexto da programação Python, quando você deseja verificar se uma string começa com uma determinada substring, qual dos seguintes padrões de código você utilizaria para realizar essa verificação?

A) if string.startsWith(substring):
B) if substring in string[0]:
C) if string.is_start(substring):
D) if string.startswith(substring):
E) if string.match(substring):

**7.** Em Python, como você expressaria um loop 'for' que itera através de uma lista chamada 'legumes'?

A) for x in legumes:
B) for x < len(legumes):
C) para x em legumes:
D) for x = 0; x < legumes.length(); x++:
E) for legumes in x:

**8.** Qual das alternativas abaixo descreve corretamente a funcionalidade do operador 'not' em Python?

A) Inverte o valor booleano de uma expressão.
B) Verifica se duas variáveis são diferentes.
C) Verifica se uma variável não está em uma lista.
D) Compara se uma variável é menor que outra.
E) Combina duas expressões booleanas.

**9.** Em Python, como você acessa o valor associado a uma chave em um dicionário?

A) dict.chave
B) dict->chave
C) dict[chave]
D) dict[chave()]
E) dict(chave)

**10.** Considere que você está implementando uma função em Python para verificar se um determinado número é par ou ímpar. Qual das opções abaixo representa corretamente a expressão condicional que você deve utilizar dentro da função?

A) if numero % 2 == 0:
B) if numero % 2 != 0:
C) if numero % 2 == 1:
D) if numero % 2 == 2:
E) if numero % 2 == 1:

**GABARITOS E COMENTÁRIOS**

**1.** Qual é a saída do seguinte código em Python?

```
x = [1, 2, 3]
x.append([4, 5])
print(len(x))
```

A) 3
B) 4
C) 5
D) 6
E) 7

**Gabarito:** B

**Comentários:** A função 'append()' adiciona um elemento ao final da lista. Neste caso, a lista '[4, 5]' é adicionada como um único elemento, aumentando o comprimento da lista 'x' para 4.

**2.** Em Python, qual é a função principal do operador '=='?

A) Atribuir valores a variáveis.
B) Concatenar duas strings.
C) Verificar se uma variável é maior que outra.
D) Comparar duas variáveis para verificar se são iguais.
E) Dividir dois números inteiros.

**Gabarito:** D

**Comentários:** O operador '==' é usado para comparar duas variáveis ou valores para verificar se eles são iguais, retornando True ou False.

**3.** No Python, qual é o resultado da seguinte expressão: 18 % 7?

A) 0
B) 1
C) 2
D) 3
E) 4

**Gabarito:** E

**Comentários:** O operador '%' em Python calcula o resto da divisão de um número por outro. No caso de 18 dividido por 7, o resultado é 2 e o resto é 4. Portanto a resposta correta é a alternativa E, 4.

**4.** Quando se utiliza um dicionário em Python, é possível obter uma lista de todas as chaves presentes no dicionário. Qual das opções a seguir demonstra corretamente como fazer isso utilizando o método apropriado?

A) chaves = dict.chaves()
B) chaves = dict.values()
C) chaves = list(dict)
D) chaves = dict.items()
E) chaves = dict.keys()

**Gabarito:** E

**Comentários:** O método 'keys()' de um dicionário em Python retorna um objeto view contendo todas as chaves do dicionário, o que permite iterar ou convertê-las em uma lista.

**5.** Considere o seguinte trecho de código em Python: 'minha_lista = [1, 2, 3, 4, 5]; nova_lista = [x*2 for x in minha_lista]'. Qual será o conteúdo da variável 'nova_lista' após a execução deste código?

A) [1, 2, 3, 4, 5]
B) [2, 4, 6, 8, 10]
C) [3, 6, 9, 12, 15]
D) [4, 8, 12, 16, 20]
E) [5, 10, 15, 20, 25]

**Gabarito:** B

**Comentários:** A compreensão de listas em Python permite criar novas listas aplicando uma expressão a cada item da lista original. Neste caso, cada elemento de 'minha_lista' é multiplicado por 2.

**6.** No contexto da programação Python, quando você deseja verificar se uma string começa com uma determinada substring, qual dos seguintes padrões de código você utilizaria para realizar essa verificação?

A) if string.startsWith(substring):
B) if substring in string[0]:
C) if string.is_start(substring):
D) if string.startswith(substring):
E) if string.match(substring):

**Gabarito:** D

**Comentários:** O método correto em Python para verificar se uma string começa com uma determinada substring é 'startswith'.

**7.** Em Python, como você expressaria um loop 'for' que itera através de uma lista chamada 'legumes'?

A) for x in legumes:
B) for x < len(legumes):
C) para x em legumes:
D) for x = 0; x < legumes.length(); x++:
E) for legumes in x:

**Gabarito:** A

**Comentários:** A resposta correta é A) for x in legumes:, porque em Python, a sintaxe de 'for' é usada para iterar sobre uma sequência (que pode ser uma lista, uma tupla, um dicionário, um conjunto ou uma string). No caso de uma lista como 'legumes', podemos iterar através de seus elementos usando a sintaxe 'for x in legumes:'. As outras alternativas não são sintaxes válidas em Python.

**8.** Qual das alternativas abaixo descreve corretamente a funcionalidade do operador 'not' em Python?

A) Inverte o valor booleano de uma expressão.
B) Verifica se duas variáveis são diferentes.
C) Verifica se uma variável não está em uma lista.
D) Compara se uma variável é menor que outra.
E) Combina duas expressões booleanas.

**Gabarito:** A

**Comentários:** O operador 'not' inverte o valor booleano de uma expressão, transformando True em False e vice-versa.

**9.** Em Python, como você acessa o valor associado a uma chave em um dicionário?

A) dict.chave
B) dict->chave
C) dict[chave]

D) dict[chave()]
E) dict(chave)

**Gabarito:** C

**Comentários:** Em Python, o valor associado a uma chave em um dicionário é acessado usando a notação de colchetes, como dict[chave].

**10.** Considere que você está implementando uma função em Python para verificar se um determinado número é par ou ímpar. Qual das opções abaixo representa corretamente a expressão condicional que você deve utilizar dentro da função?

A) if numero % 2 == 0:
B) if numero % 2 != 0:
C) if numero % 2 == 1:
D) if numero % 2 == 2:
E) if numero % 2 == 1:

**Gabarito:** A

**Comentários:** Para verificar se um número é par, você deve utilizar a expressão 'numero % 2 == 0', pois um número é par quando o resto da divisão por 2 é igual a 0.

| | | | | |
|---|---|---|---|---|
| 1.B | 2.D | 3.E | 4.E | 5.B |
| 6.D | 7.A | 8.A | 9.C | 10.A |

# ESSA LEI TODO MUNDO CONHECE:
# PIRATARIA É CRIME.

**Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.**

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

**Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.**