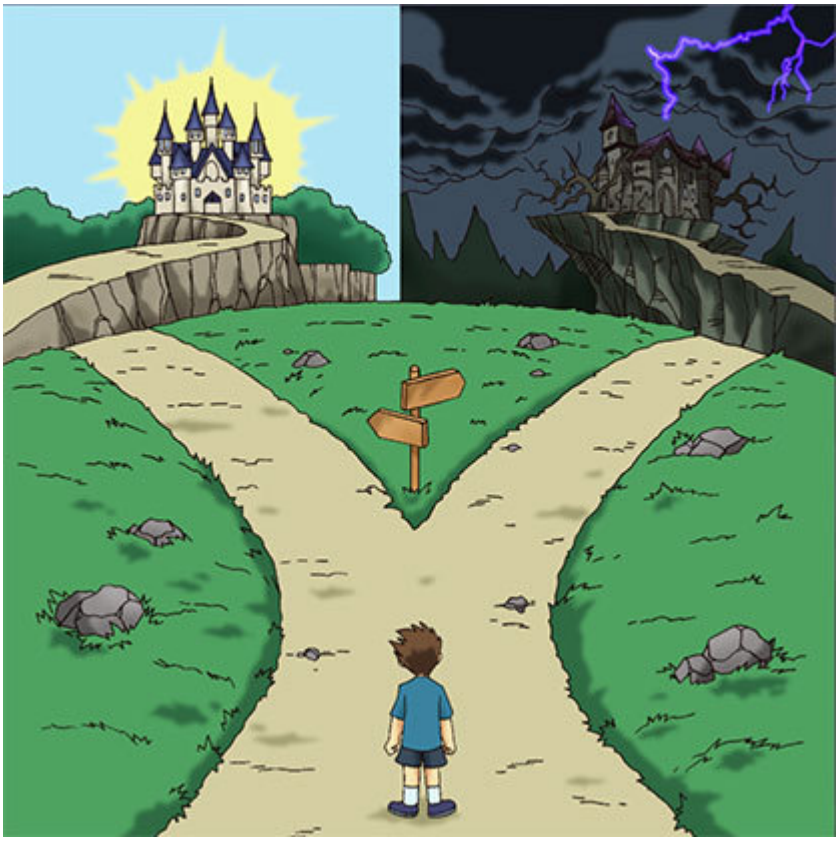
Aula 07 - Estruturas de decisão



Até o momento, falamos de instruções bastante epecíficas, que tratam a solução como um caminho único. Mas sabemos que imprevistos acontecem. E se ocorrrer do computador precisar tomar uma decisão? Como ele faria isso? Como falado anteriormente, o computador é a coisa mais estúpida que existe no universo. Ele não fará nada, a não ser que você mande ou ensine. Nesse caso, **é sua tarefa ensinar o computador a tomar decisões**.

O que é uma Estrutura de decisão?

É um bloco de programação que divide o seu código em dois algoritmos distintos, mas somente um deles será executado com base em uma condição determinada pelo próprio programador: caso a condição seja **verdadeira**, ele irá executar um algoritmo. Caso essa mesma condição seja **falsa**, um outro algoritmo será executado no lugar. Dessa forma, o programa poderá ser finalizado de duas formas diferentes. Ambos os algoritmos serão definidos pelo programador. Há também a possibilidade de ter mais de dois caminhos diferentes, e pode acontecer ainda do computador precisar avaliar duas ou mais condições diferentes simultaneamente. Entenda que é a partir daqui que iremos precisar dos operadores relacionais vistos na aula passada.

Estrutura if...else

De todas as estruturas que existem em programação, a mais famosa (e usada) é, de longe, o **if...else**, também conhecido como **se...senão**. Se você usa o Excel, então deve estar familiarizado com essa estrutura: ela é usada na função `=SE(condição;valor_se_verdadeiro;valor_se_falso)` do Excel, e segue a mesma linha de raciocínio.

Funciona assim: suponha que um programa precisa verificar se um determinado usuário de nome *Fulano* e idade *20* seja maior de idade ou não. Então o dev irá desenvolver a seguinte estrutura:

In [ ]:
```
# variáveis
nome = 'Fulano'
idade = 20

# estrutura de decisão
if idade >= 18:
    print(f'{nome} é maior de idade.')
else:
    print(f'{nome} é menor de idade.')
```

```
Fulano é maior de idade.
```

No caso acima, o programa irá executar apenas a primeira saída de dados, e ignorar a segunda, já que ela satisfez a condição imposta. Agora vamos trocar a idade do Fulano para 15 e rodar de novo o programa:

In [ ]:
```
# variáveis
nome = 'Fulano'
idade = 15

# estrutura de decisão
if idade >= 18:
    print(f'{nome} é maior de idade.')
else:
    print(f'{nome} é menor de idade.')
```

```
Fulano é menor de idade.
```

Veja que agora o resultado final foi diferente: ele ignorou a primeira saída de dados, e executou a segunda, já que a condição imposta não foi atendida.

Mais uma coisa: repare que as linhas de comando das saídas de dados estão com um recuo mais para a direita. **Este é um dos detalhes mais importantes do Python**, e que veremos à seguir.

Indentação

Indentação de código é uma prática que consiste em organizar o seu código para deixá-lo mais organizado, e consequentemente, mais legível e mais fácil para manutenção. A técnica é adicionar um espaçamento, ou um recuo à esquerda das linhas de comando que ficam dentro das estruturas, e dessa forma podemos identificar mais facilmente o que está dentro de um escopo e o que está fora dele. O problema é que essa é uma

prática que não é seguida por todos, o que acaba resultando em códigos confusos e problemáticos na hora de dar manutenção.

Python resolveu muito bem esse problema, pois nessa linguagem em questão, **a indentação é obrigatória**, já que é ela que identifica se um trecho de código está ou não dentro de uma estrutura, diferentemente das outras linguagens, onde o que define os limites da estrutura são as chaves (`{}`). Em outras palavras: deixe de indentar o seu código Python ou indente incorretamente, e é garantido que seu código dará errado. Veja o exemplo abaixo:

`if idade >= 18:`
4x espaço ou 1x tab `print(f'{nome} é maior de idade.')`
`else:`
4x espaço ou 1x tab `print(f'{nome} é menor de idade.')`

Obs: se o seu editor ou IDE estiver corretamente configurado, ao dar `Enter` no início de uma estrutura, ele irá indentar automaticamente. Veja se não é o seu caso enquanto estiver codificando:

`if idade >= 18:` Enter
indentação automática `print(f'{nome} é maior de idade.')`
`else:` Enter
indentação automática `print(f'{nome} é menor de idade.')`

Juntando estrutura de decisão com operadores booleanos

Há alguns casos em que será necessário que duas condições ou mais sejam aceitas para entrar em um resultado positivo. Vamos montar um outro cenário onde precisaríamos desse tipo de algoritmo.

Suponha que João, que possui 11 anos, e 1,25 metros de altura, está em um parque de diversões junto com seus dois irmãos: o José, que tem 13 anos e 1,30 metros de altura, e a Maria, que tem 12 anos, e 1,15 metros de altura, e um dos brinquedos, o Trem Fantasma, tem um sistema que verifica a idade e a altura do usuário. Para ter sua entrada autorizada, ele precisa ter no mínimo 12 anos **E** 1,20 metros de altura. Veja bem: as duas condições precisam ser atendidas simultaneamente. O primeiro a entrar será o João, e para verificar se ele atende aos requisitos do brinquedo, o sistema deverá ter um algoritmo semelhante a este logo abaixo:

In [ ]:
```
# variáveis
nome = 'João'
idade = 11
altura = 1.25

# verifica se o usuário possui os requisitos mínimos
if idade >= 12 and altura >= 1.2:
    print(f'A entrada de {nome} está autorizada.')
else:
    print(f'A entrada de {nome} não está autorizada.')
```

```
A entrada de João não está autorizada.
```

Veja que João não teve a sua entrada autorizada, pois não atende aos dois requisitos, mas apenas a um deles. Vamos ver agora a Maria:

In [ ]:
```
nome = 'Maria'
idade = 12
altura = 1.15

if idade >= 12 and altura >= 1.2:
    print(f'A entrada de {nome} está autorizada.')
else:
    print(f'A entrada de {nome} não está autorizada.')
```

```
A entrada de Maria não está autorizada.
```

A entrada de Maria também não está autorizada, pois embora atenda ao primeiro requisito, ela não atende ao segundo. Vejamos agora o José:

In [ ]:
```
nome = 'José'
idade = 13
altura = 1.3

if idade >= 12 and altura >= 1.2:
    print(f'A entrada de {nome} está autorizada.')
else:
    print(f'A entrada de {nome} não está autorizada.')
```

```
A entrada de José está autorizada.
```

A entrada de José foi autorizada, já que foi o único dos 3 que atendeu a todos os requisitos do brinquedo.

O comando elif

Foi falado mais cedo que é possível encadear mais do que dois caminhos em uma estrutura de if...else. Caso isso seja necessário, enquanto que em outras linguagens se utiliza o comando `else if`, em Python ele é substituído por `elif`. Vejamos um exemplo prático: digamos que um aluno chamado Fulano tirou média 10 na nota final da escola, o Cicrano tirou 6, e o Beltrano tirou 4. Nessa escola, a média para aprovação é 7, mas o aluno só vai para recuperação caso a nota dele tenha ficado entre 5 e 7. Abaixo disso, ele é sumariamente reprovado. Então o sistema vai avaliar primeiro a nota do Fulano em um algoritmo parecido com esse:

In [ ]:
```
nome = 'Fulano'
nota = 10

if nota >= 7:
    print(f'{nome} foi aprovado.')
elif nota >= 5:
    print(f'{nome} está de recuperação.')
else:
    print(f'{nome} está reprovado.')
```

```
Fulano foi aprovado.
```

Vejam que o Fulano foi aprovado, pois sua nota era maior que a mínima necessária para aprovação. Vejamos agora a nota do Cicrano:

In [ ]:
```
nome = 'Cicrano'
nota = 6

if nota >= 7:
    print(f'{nome} foi aprovado.')
elif nota >= 5:
    print(f'{nome} está de recuperação.')
else:
    print(f'{nome} está reprovado.')
```

```
Cicrano está de recuperação.
```

Cicrano acabou ficando de recuperação, pois sua nota era menor que a mínima para aprovação, porém era maior que o limite da reprovação. É isso que o `elif` faz: adiciona uma segunda verificação caso a primeira seja falsa. Dessa forma, se o meu código entrar dentro da verificação do `elif`, é porque com certeza sua nota é menor que 7, logo não terá risco de conflitos. Agora, vamos verificar a nota do Beltrano:

In [ ]:
```
nome = 'Beltrano'
nota = 4

if nota >= 7:
    print(f'{nome} foi aprovado.')
elif nota >= 5:
    print(f'{nome} está de recuperação.')
else:
    print(f'{nome} está reprovado.')
```

```
Beltrano está reprovado.
```

Beltrano não conseguiu sequer atingir a nota mínima para ao menos conseguir a recuperação. Logo, a nota dele entrou na última verificação: aquela "dos que sobram".

Operador ternário

Mais frequentemente usada por programadores mais avançados, tipo Pleno e Senior, esse tipo de estrutura é uma versão mais simplificada do mesmo `if...else` tradicional. Caso a estrutura seja simples, compensa reduzi-la para uma única linha de comando. Pegaremos como exemplo o mesmo caso da verificação de idade do início da aula, para que possamos fazer uma comparação:

In [ ]:
```
# variáveis
nome = 'Alex'
idade = 39

# operador ternário
print(nome, 'é maior de idade.' if idade >= 18 else 'é menor de idade.')
```

```
Alex é maior de idade.
```

Estrutura match...case

A estrutura `match...case` foi introduzida em versões recentes do Python, mais precisamente a 3.10.

Quem já tem experiência com programação em outra linguagem deve se lembrar que em outras linguagens de programação temos uma outra estrutura de decisão muito parecida com a `if...else`, que é a `switch...case`. Acontece que o Python **NÃO TEM SWITCH...CASE**. Ou pelo menos não tinha até a atualização 3.10. Antes dela, quem quisesse usar um `switch...case` da vida, teria que apelar para o `if...elif...else` tradicional ou usar alguma função maluca. Mas no Python 3.10 foi introduzido o `match...case`, que possui exatamente a mesma funcionalidade do `switch...case`.

Tá, mais e daí? Eu não sei o que é switch...case nem o que é match...case

O `match...case`, que é o equivalente ao `switch...case` nas outras linguagens, é uma estrutura de decisão que permite adicionar vários caminhos diferentes para o algoritmo (muito mais do que apenas 2) e que permite avaliar valores bem específicos para decidir qual caminho tomar. Ele acaba sendo um algoritmo mais ideal para alguns casos específicos, como por exemplo, quando eu tiver várias possibilidades de caminhos diferentes, ou quando as comparações a serem feitas são valores absolutos, ao invés de intervalos de números. Aqui, ele não compara usando os operadores relaccionais, mas sim verifica valor por valor, um por um, e retorna um único valor quando ele não conseguir encontrar um valor exato.

Vamos para um exemplo prático: eu vou para o cinema, mas antes de escolher o filme, gostaria de saber qual filme está passando em qual sala. Digamos que o cinema em questão tem 5 salas. Se eu selecionar um número dentro do número de salas existentes, o programa deverá me retornar o nome do filme que está passando na sala escolhida. Segue o algoritmo abaixo:

In [ ]:
```python
# entrada de dados
sala = int(input('Informe o número da sala de 1 a 5:'))

# verifica a sala
match sala:
    case 1:
        print('Filme: A Volta dos Que Não Foram.')
    case 2:
        print('Filme A Roda Quadrada.')
    case 3:
        print('Filme Poeira em Alto Mar.')
    case 4:
        print('Filme As Tranças do Rei Careca.')
    case 5:
        print('Filme A Vingança do Peixe Frito.')
    case _:
        print('Sala inexistente.')
```

```
Filme Poeira em Alto Mar.
```

O `match` faz o papel do comando `switch` das outras linguagens. A última opção, que nas outras linguagens era chamado de `default`, aqui usamos o `case _:` no lugar. Repare também que não usamos o comando `break`, usado para interromper um case. O comando `break` até existe no Python, mas não é utilizado aqui, e também não é para essa finalidade (veremos sobre ele nas aulas futuras). Aqui simplesmente não há a necessidade de colocar o comando `break`, nem de substituí-lo, já que o `break` aqui é feito de forma automática.

Estrutura try...except

Para os que já têm experiência com programação, a estrutura do `try...except` é a mesma do `try...catch` das outras linguagens de programação tradicionais. Para os que estão vendo programação pela primeira vez, o `try...except` é uma estrutura de decisão usada para **tratamento de exceções**. Ela funciona de forma parecida com a estrutura `if...else`, com a diferença de que ele irá "tentar" executar o bloco do `try`, independente de condição ou não, mas caso esse bloco, por algum motivo, der algum erro, ele para e executa outro algoritmo, o do `except`. Esse algoritmo é muito usado para evitar que o usuário veja mensagens muito técnicas de erros no programa, e retorne uma mensagem de erro mais amigável no lugar. Segue abaixo um exemplo:

In [ ]:
```
valor1 = '10' # string
valor2 = 10 # int

try:
    print(valor1 + valor2)
except:
    print('Não foi possível rodar o programa. Desculpe!')
```

```
Não foi possível rodar o programa. Desculpe!
```

No caso acima, o programa retorna a mensagem do `except`, pois não foi possível executar o `try`, visto que ele está tentando somar uma string com um número inteiro, o que é impossível para o Python. O algoritmo de baixo é o mesmo algoritmo, porém adicionado uma nova instrução para exibir uma mensagem independente de erro no programa ou não:

In [ ]:
```
valor1 = '10' # string
valor2 = 10 # int

try:
    print(valor1 + valor2)
except:
    print('Não foi possível rodar o programa. Desculpe!')
finally:
    print('Tente mudar o valor da variável "valor1" para um número inteiro caso
```

```
Não foi possível rodar o programa. Desculpe!
Tente mudar o valor da variável "valor1" para um número inteiro caso dê erro.
```

Nesse novo algoritmo, ele retorna o erro junto com uma sugestão para que o programa dê certo. Isso acontece porque a mensagem da sugestão foi programada dentro da

estrutura `finally`. Essa estrutura executa seu algoritmo independentemente do algoritmo do `try` ter dado certo ou não.

Meus parabéns!!! Agora você sabe criar estruturas de decisão em Python!

Exercícios

---

1. Crie um programa que calcule o **Índice de Massa Corporal (IMC)** do usuário, mostre o valor do IMC na tela, e retorne se o usuário está no peso ideal ou se precisa emagrecer ou engordar mais. Utilize a tabela do IMC para definir os valores.

In [ ]:
```
# TODO
```

2. Um elevador de carga possui capacidade para 200 kg. Crie um programa que receba do usuário o peso da carga, e verifica se a carga está autorizada a usar o elevador ou não.

In [ ]:
```
# TODO
```