Aula 08 - Laços de repetição

A essa altura do campeonato, você já deve ter visto que os programas só rodam uma única vez. Caso o usuário deseje rodar o programa de novo, ele deverá executar mais uma vez o programa, o que torna a tarefa de executar um programa cansativo, visto que o usuário deverá rodar várias vezes o mesmo programa apenas para testar diferentes resultados. É aí onde entram os ***loops***, ou **laços de repetição**.

Eles são blocos de programação que executam um único algoritmo várias vezes seguidas antes de finalizar. O número de vezes em que ele é executado depende de vários fatores, como vamos aprender nessa aula.

Laço while

O `while` é um bloco que repete um algoritmo enquanto uma determinada condição for verdadeira. Vamos começar com uma demonstração bem simples: criar um programa que exibe uma contagem de 10 a 0...coisa simples.

In [ ]:
```
# declaração de variável
numero = 10

# loop
while numero >= 0:
    print(numero)
    numero -= 1
```

```
10
9
8
7
6
5
4
3
2
1
0
```

Repare que com um único `print()` foram impressos 11 números, de 10 a 0. O que ele fez foi imprimir o valor da variável `numero` enquanto ela fosse maior ou igual a 0. No momento que ela chegou ao valor -1, o loop se encerrou, pois a condição não foi mais atendida. Repare também que, para que a repetição chegasse ao fim, foi necessário acrescentar o comando `numero -= 1`, que tem a função de subtrair o número 1 ao valor da variável `numero`, alterando assim o valor da variável para cada nova repetição. Caso isso não fosse feito, o programa iria exibir o número 10 em sequências intermináveis, pois o valor da variável `numero` nunca iria mudar, e consequentemente nunca iria chegar a ser menor que 0, o que faria o programa entrar no que chamamos de

loop infinito, e isso é um problema, pois o programa ficaria rodando eternamente até ocupar 100% da memória RAM e da CPU, travando assim seu computador e forçando uma reinicialização que poderia prejudicar as tarefas em execução no seu SO, ou no pior dos casos, danificar fisicamente seu PC. Outro ponto também é que o valor da variável `numero` é positivo, e foi isso que permitiu que o loop fosse executado. Caso o valor de `numero` fosse negativo, o programa não iria executar o loop.

Assim como os blocos das estruturas de decisão, o while começa e termina onde o código estiver **indentado**. Portanto, nunca se esqueça de indentar corretamente seu código, já que toda a parte do código indentada fará parte do loop.

Continue

O comando `continue` serve para que a execução do restante do bloco seja desconsiderada e o código ignorado. O loop terá sua contagem contabilizada e o programa retorna para o início do loop. Segue o exemplo abaixo:

In [ ]:
```
cont = 0

while cont < 10:
    cont += 1
    if cont % 2 == 0:
        print(cont)
    else:
        continue

    print('Contando...')
```

```
2
Contando...
4
Contando...
6
Contando...
8
Contando...
10
Contando...
```

Break

O comando `break` faz o contrário: encerra imediatamente o loop e vai para o final do programa, sem terminar o loop, mesmo que a condição esteja sendo atendida:

In [ ]:
```
cont = 0

while cont < 15:
    cont += 1
    if cont % 2 == 0:
        print(cont)
    elif cont >= 7:
        break
    else:
        continue
```

```
    print('Contando...')
```

```
2
Contando...
4
Contando...
6
Contando...
```

Pass

O comando `pass` é utilizado para o programa simplesmente pular uma determinada opção. Pode ser substituído por `...`. Veja o exemplo:

In [ ]:
```
cont = 0

while cont < 20:
    cont += 1
    if cont % 2 == 0:
        print(cont)
    elif cont < 5:
        pass
    elif cont >= 15:
        break
    else:
        continue

    print('Contando...')
```

```
Contando...
2
Contando...
Contando...
4
Contando...
6
Contando...
8
Contando...
10
Contando...
12
Contando...
14
Contando...
```

Agora o mesmo exemplo, só que com `...` no lugar do `pass`:

In [ ]:
```
cont = 0

while cont < 20:
    cont += 1
    if cont % 2 == 0:
        print(cont)
    elif cont < 5:
        ...
    elif cont >= 15:
        break
```

```
    else:
        continue

    print('Contando...')
```

```
Contando...
2
Contando...
Contando...
4
Contando...
6
Contando...
8
Contando...
10
Contando...
12
Contando...
14
Contando...
```

Laço while True

Agora uma surpresa que poderá deixar os programadores das outras linguagens chocados: **O PYTHON NÃO POSSUI O LAÇO** `DO...WHILE`. É isso mesmo: não se faz o `do...while` no Python. No lugar dele, o mesmo `while` é usado em conjunto com um valor booleano para executar um loop infinito, ou até que ocorra uma quebra no loop, como iremos aprender a seguir. Vamos retornar a um exemplo feito nas aulas anteriores para fins de demonstração: retornaremos ao algoritmo da verificação da idade, e refazê-lo para que o usuário decida quando encerrar o programa:

In [ ]:
```
# loop
while True:
    # entrada do nome
    nome = input('Informe o seu nome ou deixe em branco para sair: ')

    # verifica se o valor foi informado ou não
    if nome != '':
        # entrada da idade
        idade = int(input('Informe sua idade: '))

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

        continue # retorna ao início do loop
    else:
        break # encerra o loop
```

```
Alex é maior de idade.
Alex Machado é menor de idade.
```

No caso do algoritmo acima, o usuário informa o nome e a idade, e o programa verifica se o usuário é maior ou menor de idade. Até aí, nada diferente do que fizemos na aula

passada. Mas após fazer a verificação da idade do usuário, o comando `continue` é executado. Esse comando desconsidera todo o programa após essa linha, e retorna para o início do loop, perguntando o nome novamente. Caso o usuário deseje enncerrar o programa, basta deixar esse campo em branco, e ele irá executar o comando `break`, que simplesmente encerra o lopp e parte para o final do programa.

O legal é que essa estrutura pode ser aplicada para qualquer programa, tanto os que nós criamos nas aulas anteriores como nos próximos também.

Laço for

O laço `for` é um dos loops mais usados na programação. É um laço de repetição como o anterior também, mas com uma diferença: é sempre executado com um número finito de vezes, evitando que o programa entre em loop infinito. A desvantagem em relação so `while` é que o número de loops dessa vez é ditado pelo próprio programa ao invés do usuário final. Este, por sua vez, não possui controle sobre o número de loops a ser executado no for, ou pelo menos não diretamente. Para fins de exemplo, vaos executar um programa parecido com o primeiro programa dessa aula, mas desta vez, o programa irá exibir números de 0 a 4. Segue o algoritmo:

In [ ]:
```
for n in range(5):
    print(n)
```

```
0
1
2
3
4
```

Vamos analisar esse algoritmo: repare como ele é bem mais simplificado que o laço `while`. A variável `n` foi declarada dentro do próprio laço sem a necessidade de inicialização. Isso porque quando uma variável é declarada diretamente no laço `for`, ele automaticamente assume o valor `0` (há um motivo para isso que será explicado mais para frente). A função `range()` indica o número de loops que o laço irá obrigatoriamente executar, nesse caso serão 5 vezes. Não só isso: para cada loop, ele automaticamente irá somar mais um na variável `n` de forma automática, sem a necessidade de atribuir uma soma à variável, como ocorre no laço `while`. Dessa forma, o valor de `n` muda a cada novo loop, e não há a necessidade de colocar o comando `n += 1`.

Sobre a contagem de números decimais do computador

A essa altura do campeonato, já se sabe que o computador não executa certas tarefas da mesma forma que nós, e o mesmo vale para a contagem numérica. Para entendermos isso, precisamos entender como funciona o sistema decimal de contagem. O sistema decimal é composto pelos números de 0 a 9, e a partir disso, adicionamos mais uma casa decimal. Por exemplo: o número após o 9 é o 10, com duas casas decimais. No mundo real, o ser humano utiliza esse sistema para contar quantidade de elementos. Por

exemplo: 0 maçâs significa nenhuma maçã, e a partir daí temos 1 maçã, 2 maçãs, 3 maçãs....e por aí vai.

O computador entende esse sistema de forma diferente, já que ele começa a contagem a partir do número 0, e não do 1. É por isso que a contagem no algoritmo anterior começou do 0 e parou no 4. O primeiro número foi 0, e o quinto número foi o 4. Se o `range()` tivesse o número 8, a contagem ia ser até o número 7, e assim por diante. Este detalhe será de extrema importância para a próxima aula, já que o laço `for` é muito utilizado para trabalhar com listas, que é exatamente o assunto da próxima aula.

Exercícios

---

1. Crie um programa que receba 2 números do usuário, e que ele possa escolher um das 4 operações matemáticas para calcular os dois números. O programa deverá exibir na tela o resultado da conta matemática. O programa deverá dar também ao usuário uma opção para sair do programa caso deseje, e que ele possa fazer quantos cálculos desejar.

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

2. Crie um programa que liste 5 salas de cinema, e mostre os filmes de cada uma das salas e suas classificações indicativas. O usuário deverá informar sua idade e a sala com o filme desejado. Caso o usuário tenha a idade mínima para ver o filme, o programa irá imprimir o ingresso para o filme desejado. Caso o usuário não tenha a idade mínima, o programa deverá informar que ele não tem idade para ver o filme, e deverá retornar a lista de filmes para que o usuário escolha outro filme.

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