Resumo - Aula 3 - while (repetição)

💡 Nessa aula falamos sobre a instrução `while` do Python, que permite que um conjunto de instruções sejam executadas repetidas vezes **enquanto** uma expressão for verdadeira.

Sintaxe

```
while <expressão booleana>:
    <instrução 1>
    <instrução 2>
    ...
    # Depois dessa instrução (última) avalia-se a expressão novamente
    <última instrução>
print("Aqui estou fora do while!")
```

Onde `<expressão booleana>` pode ser qualquer expressão que resulte em um valor verdadeiro ou falso (`True` ou `False`). Por exemplo:

```
x = 10
while(x <= 20):  # x <= 20 vai ser "avaliado" a cada ciclo
    print(x)
    x = x + 1  # Incrementa o valor de x em 1
    # Quando x = 20 e incrementarmos +1, ele vai passar a ser 21
    # e x <= 20 vai ser False.
```

💡 Chamamos de **iteração** cada "passada" no trecho de código dentro de um `while`. No exemplo anterior, quantas iterações serão realizadas?

💡 **loop infinito**: quando o programa fica infinitamente executando um conjunto de instruções. A causa mais comum é quando a `expressão booleana` sempre retorna `True`. Por exemplo, e se não incrementássemos o valor de `x` no exemplo anterior? `x` seria sempre igual a `10` e **sempre** `<= 20`.

Break

A instrução `break` faz com que o programa "saia" de dentro do loop. Por exemplo:

```
while(True):  # Sempre é verdadeira essa condição
    break
    print("Não vai imprimir isso na tela")
print("Mas isso vai.")
```

Execute o programa acima para ver o que acontece. Depois tente tirar o `break` e veja o que acontece.

💡 Lembre-se: você pode apertar `Ctrl + C` para parar a execução de um programa no Python Shell.

Exemplos

Somar todos os números de 1 a n

```
n = int(input("Digite um número: "))
i = 1
soma = 0
while(i <= n):
    soma = soma + i  # Seria a mesma coisa escrever soma += i
```

```
    i += 1  # Precisamos incrementar i senão teríamos um loop infinito
    # Ou: i = i + 1, exatamente a mesma coisa
print("A soma é: ", soma)
```

Acerte um número

Enunciado: escreva um programa que pede para o usuário tentar acertar um número (`resposta`). Se ele errar 3 vezes, ele perde. Caso ele acerte antes disso, ele ganha.

```
resposta = 9
tentativas = 3  # máximo de tentativas
i = 0  # Nosso contador de quantos chutes o usuário já deu

# Observe que inicializamos i como 0  e tentativas como 3
# Logo, temos que usar i < tentativas, e não i <= tentativas.
# Tente entender o que aconteceria caso utilizássemos "menor ou igual" ao
# invés de simplesmente "menor".

# Enquanto o número de chutes for menor que o máximo de tentativas
while(i < tentativas):
    chute = int(input("Tente acertar o número: "))
    if chute == resposta:
        print("Você acertou, parabéns!")
        break
    else:
        print("Você errou!")
        i = i + 1
        print("Você tem {} tentativas".format(tentativas - i))

print("Fim!")
```

Exercícios (dever de casa)

1. **FizzBuzz**. Refaça o exercício do FizzBuzz mas agora solicite um número para o usuário e *itere* (percorra) todos os números de 1 até `n` (onde `n` é o número que o usuário digitar).
2. **Número primo.** Faça um programa que receba um número do usuário (`n`) e diga se ele é um número primo ou não. Lembre-se: um número é primo quando é **divisível** por 1 e por ele mesmo.
    1. **Bônus**: escreva outro programa onde o usuário digita um número e o programa imprime **todos os números primos de 1 até N** (onde `N` é o número que o usuário digitou).
3. **Fatorial**. O fatorial de um número `n` é definido da seguinte maneira:

    `fatorial(n) = n * (n-1) * (n-2) * ... * 1`

    Ou seja: `fatorial(5) = 5 * 4 * 3 * 2 * 1 = 120`

    Escreva um programa que receba um número e imprima o seu fatorial. **Lembre-se: fatorial(0) = 1**.

    Dica: assim como utilizamos contadores indo de 1 até um certo limite, podemos utilizar contadores que **diminuem**, ou seja, começam em um certo número `n` e vão diminuindo até `0`, por exemplo. (`i = i - 1` vs `i = i + 1`)