

# Paradigmas de Linguagem de Programação em Python



# Estruturas de Repetição



[repeticoes+math.ipynb](#)



## Estruturas de Repetição

Imagine um programa que calcula a média de um aluno:

```
nome = input("Nome: ")
n1 = float(input("Nota da primeira prova: "))
n2 = float(input("Nota da segunda prova: "))

media = (n1 + n2) / 2
if media >= 7:
    print("O aluno %s foi aprovado com media %.2f" % (nome, media))
else:
    print("O aluno %s foi reprovado com media %.2f" % (nome, media))
```

## Estruturas de repetição

O programa anterior recebe um nome e duas notas, e ao final informa se o aluno foi aprovado ou não.

Se tivermos apenas **um** aluno, o programa é plenamente satisfatório.

Se tivermos **dois** alunos, já começa a ficar meio repetitivo e ineficiente.

## Estruturas de repetição

E agora se tivermos uma turma com 60 alunos?

Como poderíamos fazer para calcular a média de todos os alunos em um mesmo programa de forma prática?

Observe que precisamos repetir os mesmos comandos, várias vezes, até que a média de todos os alunos tenha sido calculada e impressa.

Por isso, estruturas de repetição representam a base de vários programas.

## Estruturas de repetição

Ainda, note que:

1. Se o usuário sabe, de antemão, a quantidade de alunos, seria ideal repetir o código anterior automaticamente, sem que precisássemos escrevê-lo para cada aluno.
2. Se o usuário não sabe a quantidade, o código seria repetido enquanto o usuário quisesse.

## Estruturas de repetição

As linguagens de programação oferecem **mecanismos** para **repetir comandos** de forma elegante:

Os famosos **Laços** (em inglês: *Loops*) !!

Assim, veremos os dois principais comandos de repetição do Python: **while** e **for**

## Estruturas de repetição

### while

*While*, do inglês, significa "enquanto"

Enquanto algo for verdadeiro, repita os comandos!

"Enquanto **i** for menor que 6, exiba **i**"

```
i = 1
while i < 6:
    print(i)
    i += 1
```



## Estruturas de repetição

### while

E quando não soubermos a quantidade exata de repetições?

```
x = float(input(""))
while x > 0:
    print(x)
    x = float(input(""))

print("Fim da repeticao")
```

## Estruturas de repetição

### **while**

No programa anterior, enquanto o usuário entrar com valores positivos, o laço (*loop*) **while** continuará ativo.

Note que **x** foi inicializado via **input()** fora do laço.

Ou seja, a condição é baseada num critério lógico:

Enquanto a expressão lógica for **True**, continue o *loop*.

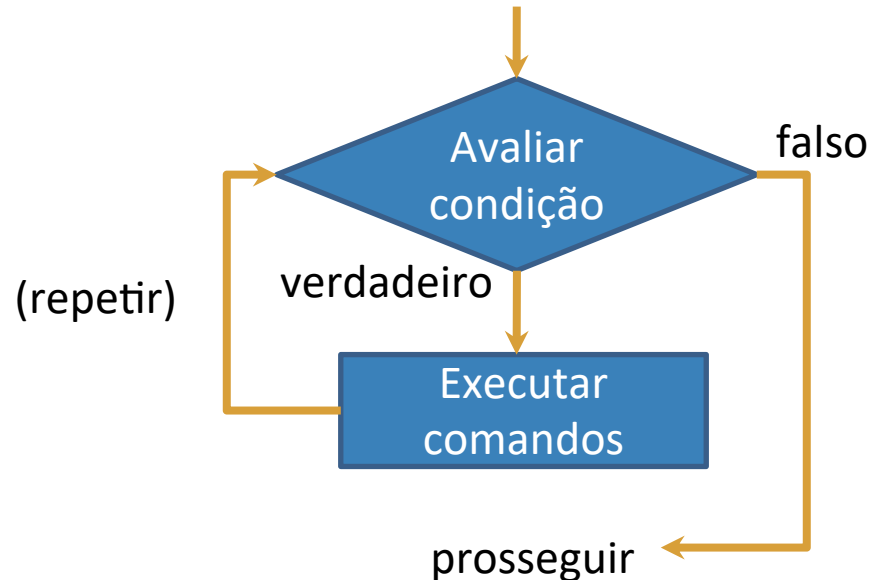
Quando ela se torna **False**, o programa sai do bloco comandos indentados.

Cada rodada do laço é chamada **iteração** (e não “interação”).

## Estruturas de repetição

### while

Note ainda, que o **while** requer que as variáveis que definem a condição do *loop* estejam prontas para uso.



## Estruturas de repetição

### while

- Imprimindo os números inteiros de 1 a 10.

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

- **i** é inicializado fora do laço e incrementado a cada iteração!

- **i += 1** ↔ **i = i + 1**

## Estruturas de repetição

### while

### Contadores

Quando precisamos **incrementar** uma variável para **fazer** algum tipo de controle, dizemos que essa variável é um contador.

No exemplo anterior, **i** é um contador

Voltando ao problema do cálculo da média dos alunos... vamos supor que **sabemos** que a turma tem **10 alunos!**

# Estruturas de repetição

## while

### Contadores

```
total = 68
cont = 1
while cont <= total:
    nome = input("Nome: ")
    n1 = float(input("Nota da primeira prova: "))
    n2 = float(input("Nota da segunda prova: "))
    media = (n1 + n2) / 2
    if media >= 7:
        print("O aluno %s foi aprovado com media %.2f" % (nome, media))
    else:
        print("O aluno %s foi reprovado com media %.2f" % (nome, media))
        cont += 1
print("Fim!")
```

## Estruturas de repetição

### while

```
i = 1
soma = 0
while i <= 10:
    x = int(input("Digite o valor do numero %d: " % i))
    soma += x
    i += 1
print("Soma: %d" % soma)
```

## Estruturas de repetição

**while**

### Acumuladores

Nem só de contadores vive o homem!

Quando precisamos **acumular** valor para resolver algum tipo de problema, dizemos que essa variável é um **acumulador**.

No exemplo anterior, **soma** é um acumulador

A diferença entre um contador e um acumulador é que nos contadores o valor adicionado é constante e, nos acumuladores, variável.



# Estruturas de repetição

## while

### Acumuladores

```
total, cont, soma = 10, 1, 0
while cont <= total:
    nome = input("Nome: ")
    n1 = float(input("Nota da primeira prova: "))
    n2 = float(input("Nota da segunda prova: "))
    media = (n1 + n2) / 2
    if media >= 7:
        print("O aluno %s foi aprovado com media %.2f" % (nome, media))
    else:
        print("O aluno %s foi reprovado com media %.2f" % (nome, media))
    soma += media
    cont += 1
print("Soma das medias = %.2f" % (soma))
```

## Estruturas de repetição

### while

E se a quantidade não é conhecida previamente?

O usuário poderá definir um "valor padrão" para informar que não há mais alunos.

- Ex.: se `nome == '-1'`, o laço será terminado.

## Estruturas de repetição

### while

```
nome = input("Nome: ")
while nome != '-1':
    n1 = float(input("Nota da primeira prova: "))
    n2 = float(input("Nota da segunda prova: "))
    media = (n1 + n2) / 2
    if media > 7:
        print("O aluno %s foi aprovado com media %.2f" % (nome, media))
    else:
        print("O aluno %s foi reprovado com media %.2f" % (nome,
media))
    nome = input("Nome: ")

print("Fim!")
```

## Estruturas de repetição

### while

- `'-1'` passa a ser um critério de parada do laço **while**
- Embora muito útil, a estrutura `while` só verifica esse critério no início de cada repetição.
- Dependendo do problema, a habilidade de terminar o *loop* antes da avaliação normal pode ser necessária.
- Existem outras maneiras de interromper a execução de um *loop* ou pular para a próxima iteração.
  - **break**
  - **continue**

## Estruturas de repetição

### break

O **break** interrompe o laço ao ser executado, mesmo se sua condição de parada seja **True**

```
while True:
    nome = input("Nome: ")
    if nome == '-1':
        break

    n1 = float(input("Nota da primeira prova: "))
    n2 = float(input("Nota da segunda prova: "))
    media = (n1 + n2) / 2
    if media > 7:
        print("O aluno %s foi aprovado com media %.2f" % (nome, media))
    else:
        print("O aluno %s foi reprovado com media %.2f" % (nome, media))
```

## Estruturas de repetição

### **continue**

- O **continue** interrompe a execução da ATUAL iteração do laço, e pula para a próxima

```
while True:
    nome = input("Nome: ")
    if len(nome) == 0:
        continue
    if nome == '-1':
        break
    n1 = float(input("Nota da primeira prova: "))
    n2 = float(input("Nota da segunda prova: "))
    media = (n1 + n2) / 2
    if media > 7:
        print("O aluno %s foi aprovado com media %.2f" % (nome, media))
    else:
        print("O aluno %s foi reprovado com media %.2f" % (nome, media))
```

## Estruturas de repetição

### Voltando pro while ...

- Note que a condição do **while** é sempre **True**.
- Logo, é preciso que haja um **break** para não cairmos num *loop* infinito.

## Estruturas de repetição

### for

*For*, do inglês, significa "para"

"Para **x** entre 1 e 9 faça... "

```
for x in range(1, 10):  
    print(x)
```



## Estruturas de repetição

### for

Quando sabemos previamente a quantidade de iterações desejada, é mais comum utilizarmos o **for**.

O **while** permite facilmente uma quantidade indefinida de iterações, enquanto **for** não foi “feito” para isso.

É comum que **for** venha associado a um comando **range()**:

Iremos precisar bastante dessa função!

## Estruturas de repetição **for**

Função **range()**:

**range(10)** cria, internamente, um conjunto de inteiros que vão de 0 a 9, com incremento de 1

**range(1, 10)** cria, internamente, um conjunto de inteiros que vão de 1 a 9, com incremento de 1

**range(1, 10, 2)** cria um conjunto de inteiros que vão de 1 a 9, com incremento de 2.

Note que o "limite máximo" é sempre um intervalo aberto

## Estruturas de repetição

### for

A função **range()** acaba trazendo um conceito de lista, e de fato ela gera uma "lista" de números a partir de limites fixos

Ou seja, o **for** acaba sendo uma ferramenta que passeia por dentro de valores de tipos de dados quaisquer...

Inclusive **strings**!

Além disso, ele não necessita que a variável de controle seja previamente inicializada, como no **while**

## Estruturas de repetição **for**

- Com **strings**

```
texto = "Ola, Mundo!"  
for c in texto:  
    print(c)
```

## Estruturas de repetição **for**

Assim como no **while** também podemos utilizar:

**break**

**continue**

## Estruturas de repetição **for**

- Assim como no **while** também podemos utilizar:
  - **break**
  - **continue**
- Mas...agora faz sentido falarmos de outro comando, o **pass**
  - O **while** dificilmente fica sem comandos, já que precisamos inserir código para controle do laço
  - Já o **for** não, já que o controle do laço está "embutido" no uso do comando

## Estruturas de repetição

### **pass**

O **pass** ajuda a evitar erros quando, por algum motivo, precisamos ter algum comando sem um bloco de instruções. Ele simplesmente passa, como o próprio nome indica.

```
for x in range(1, 10, 2):
```

```
    pass
```

```
-----
```

```
x = 0
```

```
if x > 0:
```

```
    pass
```

## Estruturas de repetição

### Laços aninhados

Assim como fazemos com comandos condicionais, podemos combinar (aninhar) vários *loops*, conforme a necessidade

```
for tabuada in range(1, 11):  
    numero = 1  
    while numero <= 10:  
        resultado = tabuada * numero  
        print("%d x %d = %d" % (tabuada, numero, resultado))  
        numero += 1
```



## math

Python tem um módulo *built-in* que pode ser usado para tarefas matemáticas:

**math**

Para usá-lo, é preciso adicioná-lo ao conjunto de ferramentas do nosso código através de uma **importação**

**import** **math**

Depois da importação, é possível utilizar todos os seus métodos, ou funções (operações)

## math Métodos

Método	Descrição
math.cos(x)	Retorna o valor do cosseno de x
math.sin(x)	Retorna o valor do seno de x
math.exp(x)	Retorna o valor de $E^x$ , onde E é o número de Euler ( $\approx 2,7182$ )
math.sqrt(x)	Retorna o valor da raiz quadrada de x
math.fabs(x)	Retorna o valor absoluto de x
math.ceil(x)	Retorna o valor de x arredondado para cima
math.floor(x)	Retorna o valor de x arredondado para baixo

## math Métodos

Método	Descrição
math. <b>factorial</b> (x)	Retorna o valor do fatorial de x
math. <b>log10</b> (x)	Retorna o valor do logaritmo de x na base 10
math. <b>radians</b> (x)	Retorna o valor x convertido para radiano
math. <b>tan</b> (x)	Retorna o valor da tangente de x
math. <b>acos</b> (x)	Retorna o valor do arco cosseno de x
math. <b>asin</b> (x)	Retorna o valor do arco seno de x
math. <b>tanh</b> (x)	Retorna o valor da tangente hiperbólica de x

## math Constantes

Constantes	Descrição
math.e	Retorna o número de Euler ( $\approx 2.7182 \dots$ )
math.pi	Retorna o valor de PI ( $\approx 3,1415\dots$ )
math.inf	Retorna o valor real (floating-point) infinito positivo
math.tau	Retorna o valor de tau ( $\approx 6,2831\dots$ )

## math Exemplos

```
import math
```

```
print(math.pi)
```

```
print(math.e)
```

```
ceil = math.ceil(2.34)
```

```
floor = math.floor(3.75)
```

```
sqrt = math.sqrt(2)
```

```
fat = math.factorial(4)
```

```
fabs = math.fabs(-45.2)
```

Obrigado!

Alguma dúvida?