



INSTITUTO
FEDERAL
Alagoas



ATLETAS DA PROGRAMAÇÃO

Aula 2

TÓPICOS DA AULA DE HOJE

- Estruturas de decisão (if, elif, else);
- Laços (contadores e acumuladores);
- Estruturas de Dados (tuplas, listas e dicionários);
- Compreensão de lista;

ESTRUTURAS DE DECISÃO

else: a palavra-chave **else** é chamada, inevitavelmente, caso as condições anteriores tenham sido falsas.

Exemplo:

```
n1 = float(input("Nota Mensal: "))
```

```
n2 = float(input("Nota Bimestral: "))
```

```
media = (n1 + n2)/2
```

```
print("Sua média é", media)
```

```
if (media >= 6):
```

```
    print("Aluno Aprovado!!")
```

```
elif (media < 6 and media >= 4):
```

```
    print("Aluno em recuperação!")
```

```
else:
```

```
    print("Aluno reprovado!")
```

ESTRUTURAS DE DECISÃO



if: uma estrutura de decisão é construída usando a palavra-chave **if**.

Ex:

```
n1 = float(input("Nota Mensal: "))
n2 = float(input("Nota Bimestral: "))
media = (n1 + n2)/2
print("Sua média é", media)
if (media >=6 ):
    print("Aluno Aprovado!!")
```

ESTRUTURAS DE DECISÃO

elif: A palavra-chave **elif** é uma maneira de dizer: "se as condições anteriores não forem verdadeiras, tente esta condição."

Ex:

```
n1 = float(input("Nota Mensal: "))
n2 = float(input("Nota Bimestral: "))
media = (n1 + n2)/2
print("Sua média é", media)
if (media >= 6):
    print("Aluno Aprovado!!")
elif (media < 6):
    print("Aluno em recuperação!")
```

LAÇOS

Loops são, como o próprio nome indica, repetições de um mesmo bloco de código.

while: com a função **while**, podemos repetir a execução de um conjunto de instruções enquanto uma condição for verdadeira.

Ex: Imprima i enquanto i for menor que 6.

```
i = 1
```

```
while i < 6:
```

```
    print(i)
```

```
    i += 1
```

Nota: lembre-se de incrementar i, senão o loop continuará para sempre.

LAÇOS

O loop while requer que as variáveis relevantes estejam prontas, neste exemplo precisamos definir uma variável de indexação, i , que definimos como 1.

- Contadores (Ex: $x+1/10/30$)
- Acumuladores (Ex: $x+valor$)

LAÇOS

for: a função **for** é usada similarmente ao **while**. A diferença, aqui, é que o **for** precisa de um número definido de loops, enquanto o **while** pode ser usado quando você desconhece a quantidade total de repetições.

Normalmente, usa-se a função **range()**, a qual chamamos de **gerador**. Ele serve de contadora, começando em 0 por padrão e incrementando em 1 (por padrão), até chegar ao final do intervalo.

Ex: usando a função **range()**.

```
for x in range(6):
```

```
    print(x)
```


LAÇOS

Podemos usar o for, também, como **iterador**. Isto é, em vez de criarmos um contador com a função `range()`, podemos percorrer os elementos de uma estrutura de dados, que é uma lista, uma tupla, um dicionário, uma string ou outros).


Sendo assim, com o loop for podemos executar um conjunto de instruções, uma vez para cada item de uma lista, tupla, etc. Além disso, não é necessário que uma variável de indexação seja definida antecipadamente, pois a nossa referência será os índices da estrutura de dados.

Ex: Imprima cada fruta em uma lista de frutas.

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    print(x)
```



ESTRUTURA DE DADOS

LISTAS: O QUE SÃO?



As listas são usadas para armazenar vários itens em uma única variável.

Elas são um dos 4 tipos de dados internos do Python usados para armazenar coleções de dados, os outros 3 são Tuple , Set e Dictionary , todos com qualidades e usos diferentes.

LISTAS: COMO CRIAR UMA LISTA

```
thislist = ["apple", "banana",  
"cherry"]  
  
print(thislist)
```

- Uso de colchetes.

LISTAS: MUTABILIDADE

Para adicionar um item ao final da lista, use o método **append()**:

```
thislist = ["apple",  
"banana", "cherry"]  
  
thislist.append("orange")  
  
print(thislist)
```

A lista é mutável, o que significa que podemos alterar, adicionar e remover itens em uma lista após ela **ter sido** criada.

LISTAS: MULTABILIDADE

Para inserir um item de lista em um índice especificado, use o método **insert()**.

O método insert() insere um item no índice especificado:

```
thislist = ["apple", "banana",  
            "cherry"]  
  
thislist.insert(1, "orange")  
  
print(thislist)
```

Ex:

Insira um item na segunda posição.

LISTAS: MUTABILIDADE

**Remover item
especificado:**

O método **remove()**
remove o item
especificado.

```
thislist = ["apple",  
            "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)
```

Ex: Remova a
"banana":

LISTAS: INDEXAÇÃO

```
list1 = ["a", "b", "c"]  
  
print(list1[1])
```

Podemos nos
referir a um item
específico na lista.



LISTAS: FATIAMENTO

Você pode selecionar um intervalo de índices especificando onde começa e onde termina o intervalo.

Ao especificar um intervalo, o valor de retorno será uma nova lista com os itens especificados.

```
thislist = ["apple",  
"banana", "cherry", "orange",  
"kiwi", "melon", "mango"]  
  
print(thislist[2:5])
```

resultado:

```
['cherry', 'orange', 'kiwi']
```

Ex: Retorne o terceiro, quarto e quinto item.

LISTAS: LISTAS ANINHADAS

Existem várias maneiras de unir ou concatenar duas ou mais listas em Python.

Uma das maneiras mais fáceis é usando o + operador.

```
list1 = ["a", "b", "c"]
```

```
list2 = [1, 2, 3]
```

```
list3 = list1 + list2
```

```
print(list3)
```

Ex:
Junte as duas listas.

LISTAS: LISTAS ANINHADAS

Ou você pode usar o método **extend()**, cuja finalidade é adicionar elementos de uma lista para outra lista.

```
list1 = ["a", "b" , "c"]
```

```
list2 = [1, 2, 3]
```

```
list1.extend(list2)
```

```
print(list1)
```

Ex: Use o método **extend()** para adicionar list2 no final de list1.

MÉTODOS BÁSICOS DE LISTAS

| Método | Parâmetros | Resultado | Descrição |
|---------|---------------|-------------|--|
| append | item | mutador | Acrescenta um novo item no final da lista |
| insert | posição, item | mutador | Insere um novo item na posição dada |
| pop | nenhum | híbrido | Remove e retorno o último item |
| pop | posição | híbrido | Remove e retorna o item da posição. |
| sort | nenhum | mutador | Ordena a lista |
| reverse | nenhum | mutador | Ordena a lista em ordem reversa |
| index | item | retorna idx | Retorna a posição da primeira ocorrência do item |
| count | item | retorna ct | Retorna o número de ocorrências do item |
| remove | item | mutador | Remove a primeira ocorrência do item |

DICIONÁRIOS



O que são dicionários x o que são mapeamentos:

Os dicionários são usados para armazenar valores de dados em pares chave:valor.

Um dicionário é uma coleção mutável e que não permite chaves duplicadas.

Eles podem ser vistos com **mapeamentos**, isto é, a partir de uma chave, que pode ser qualquer tipo primitivo, para um respectivo valor.

DICIONÁRIOS: COMO CRIAR

Os dicionários são criados com “{ }”, sendo constituídos por chaves e valores.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
print(thisdict)
```

Ex:
Crie e imprima um dicionário.

DICIONÁRIOS: COMO ACESSAR UM ITEM

Você pode acessar os itens de um dicionário consultando o nome da chave, entre colchetes.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
x = thisdict["model"]
```

Ex:
Obtenha o valor da chave "model".

DICIONÁRIOS: DICIONÁRIOS ANINHADOS

Um dicionário Python aninhado é um dicionário dentro de um dicionário, onde os valores do dicionário externo também são dicionários.

```
d1 = {  
    0: {'Dept' :  
        'Mathematics', 'Prof': 'Dr Jack'},  
    1: {'Dept' :  
        'Physics', 'Prof': 'Dr Mark'}  
}  
print(d1)
```

Resultado:

```
{0: {'Dept': 'Mathematics', 'Prof':  
    'Dr Jack'}, 1: {'Dept': 'Physics',  
    'Prof': 'Dr Mark'}}
```

Ex:
Crie e imprima um dicionário.

MÉTODOS BÁSICOS DE DICIONÁRIOS

| MÉTODO | DESCRIÇÃO |
|---|--|
| <u>DICT.CLEAR()</u> | Remove todos os elementos do dicionário |
| <u>DICT.COPY()</u> | Copia o dicionário |
| <u>DICT.GET(KEY, DEFAULT=NONE)</u> | Retorna o valor da KEY passada como parâmetro |
| <u>DICT.HAS KEY(KEY)</u> | Retorna True se a chave estiver contida no dicionário |
| <u>DICT.ITEMS()</u> | Retorna todos os elementos do dicionário no tipo (chave, valor) |
| <u>DICT.KEYS()</u> | Retorna uma lista com as chaves do dicionário |
| <u>DICT.SETDEFAULT(KEY, DEFAULT=NONE)</u> | Como o .GET(), mas fixa como default o valor da chave que não está no dicionário |
| <u>DICT.UPDATE(DICT2)</u> | Adiciona um elemento ao dicionário, onde DICT2 é um par (chave, valor) |
| <u>DICT.VALUES()</u> | Retorna uma lista com os valores do dicionário |

TUPLAS: O QUE SÃO?



Tuplas são usadas para armazenar vários itens em uma única variável.

Imutabilidade: As tuplas são imutáveis, o que significa que não podemos alterar, adicionar ou remover itens após a criação da tupla.

TUPLAS: COMO CRIAR UMA TUPLA?

Tuplas são
criadas com
parênteses.

```
thistuple = ("apple", "banana",  
"cherry")  
print(thistuple)
```

Ex:
Crie uma tupla.



ATIVIDADE PARA CASA:

Pesquisar sobre compreensão de listas.



ATÉ A PRÓXIMA
AULA!

