Aula 2 - Conceitos básicos Python

2.1 Variáveis

Python é uma linguagem de programação dinâmica, o que signica que não é necessário definir o tipo das variáveis, basta fazer a atribuição que o interpretador faz a inferencia: Abaixo 3 declarações de variáveis, uma string, um numero real (double) e uma lista, todos são declarados da mesma forma.

In [ ]:
```
v1 = "Sou uma string"
v2 = 10.8
v3 = [1,2,3]
```

In [ ]:
```
print(v3)
```

2.2 Objetos

Tudo em Python são objetos! Todo número, string, estrutura de dados, função, classe, módulo e assim por diante, existe no interpretador Python em sua própria "caixa", que é referenciada como um objeto Python. Os objetos possuem atributos e métodos internos que podem ser usadas. Para acessar, usamos o 'ponto' e o nome dos atributo ou método.

OBS: Comentários em Python são escritos com o sustenido(#). Tudo escrito após essa marcação será ignorado pelo interpretador Python

In [ ]:
```
# Função da string que conta quantas vezes a letra aparece
print(v1.count('s'))
```

Quando fazemos uma atribuição a uma variável em Python, estamos criando referência ao objeto do lado direito do sinal de igualdade, ou seja, criamos uma cópia do objeto.

In [ ]:
```
print(v3)
v4 = v3
v4.append(4)

# Alterando V4 altera v3
print(v3)
```

2.3 Referências dinâmicas, tipos fortes

Referências a objetos não possuem nenhum tipo associado à elas.

In [ ]:
```
v3 = "Eu era uma lista, agora sou uma string!"
print(v3)

# A informação sobre o tipo é armazenada no próprio objeto
print(type(v3))
```

Porém Python ainda é uma linguagem tipada (conversões automáticas não ocorrerão)

In [ ]:
```
a1 = 5
a2 = '5'
print(a1 + a2)

# Ele não faz a conversão automática de string para inteiro
```

2.4 Operadores binários e comparações

A maior parte das operações matemáticas binárias está de acordo com o esperado:

In [ ]:
```
5 - 7
```

In [ ]:
```
12 + 21.5
```

In [ ]:
```
5 <= 2
```

A tabela abaixo exemplifica os operadores em Python:

Operador	Descrição
a + b	Soma a e b
a - b	Subtrai b de a
a * b	Multiplica a por b
a / b	Divide a por b
a // b	Faz a divisão pelo piso de a por b, descartando qualquer resto fracionário
a ** b	Eleva a a potencia de b
a and b	True se tanto a quanto b forem True; se forem inteiros é a operação bit a bit
a or b	True se ou a ou b for True
a ^ b	Or exclusivo, true se a ou b forem verdadeiros, porém não os dois
a == b	True se a for igual a b
a != b	True se a for diferente de b
a < b, a <=b	True se a for menor (menor ou igual) a b
a > b, a >= b	True se a for maior (maior ou igual) a b
a is b	True se a e b referenciam o mesmo objeto
a is not b	True se a b referenciam objetos diferentes

2.5 Objetos mutáveis e imutáveis

A maioria dos objetos em Python é mutável, ou seja, pode ter seus valores alterados.

In [ ]:
```
L1 = ["Elemento 1", 2, 10]
print(L1)
L1[1] = "Era um inteiro e agora virou uma string"
print(L1)
```

Outros objetos, como strings e tuplas são imutáveis.

In [ ]:
```
s1 = "String"
print(s1[2])
s1[2] = "i"
```

2.6 Tipos escalares

Tipos numéricos

Os principais tipos para tratar números em Python são int e float (inteiro e real)

In [ ]:
```
i1 = 8    # int
f1 = 10e3 # float em formato cientifico
f3 = 10.3 # float

print(f1)
```

Strings

Muitas pessoas utilizam Python devido as suas funcionalidades para o processamento de strings. String podem ser escritas com aspas 'simples' ou aspas "duplas"

In [ ]:
```
s1 = "Uma forma de escrever uma string é esta"
s2 = 'Já outra forma é esta'
```

String que ocupam mais de uma linha podem ser escritas envoltas em aspas '''triplas'''

In [ ]:
```
s_gigante = '''Esta
                é uma string
                que ocupa 3 linhas inteiras!'''
print(s_gigante)
```

Podemos converter objetos em strings com a função str()

In [ ]:
```
n1 = 5
print(type(n1))
s_n1 = str(n1)
print(type(s_n1))
```

As strings são sequências de caracteres, de forma que podemos tratá-las como outras sequências (listas e tuplas)

In [ ]:
```
print(s_gigante[2])
print(s_gigante[2:50]) # imprime a string da posição 2 até a 50 (isso se chama fatiamento)
```

In [ ]:
```
Somar duas strings faz com que elas sejam concatenadas e uma nova string é gerada.
```

In [ ]:
```
s1 = "Ola, meu"
s2 = " nome é Baltazar!"
s3 = s1 + s2
print(s1 + s2)
```

A formatação de strings é muito importante, e existem diversas formas de se fazer a formatação.

In [ ]:
```
s1 = "Imprima o número {:e} como científico e o número {} normal".format(10,20)
print(s1)
```

Datas e horas

O módulo embutido datetime de Python disponibiliza os tipos **datetime**, **date** e **time**. O tipo datetime combina os tipos date e time.

In [ ]:
```
from datetime import datetime, date, time

# Criando um objeto do tipo datetime
dt = datetime(2011, 10, 29, 10, 25, 50)

# Acessando o dia, hora e minutos
print("dia : ",dt.day)
print("hora : ",dt.hour)
print("minuto : ",dt.minute)
```

Podemos formatar o datetime como uma string com o método **strftime**

In [ ]:
```
# o %d/%m/%Y dita o formato d:dia m:mes Y: ano, com barras invertidas
print(dt.strftime("%d/%m/%Y"))

# Formatando com um ponto, sem o ano
print(dt.strftime("%d.%m"))
```

A tabela abaixo especifica todos as formatações de datas

Diretiva	Significado	Exemplo	Notas
`%a`	Dias da semana como nomes abreviados da localidade.	Sun, Mon, ..., Sat (en_US); So, Mo, ..., Sa (de_DE)	(1)
`%A`	Dia da semana como nome completo da localidade.	Sunday, Monday, ..., Saturday (en_US); Sonntag, Montag, ..., Samstag (de_DE)	(1)
`%w`	Dia da semana como um número decimal, onde 0 é domingo e 6 é sábado.	0, 1, ..., 6	
`%d`	Dia do mês como um número decimal com zeros a esquerda.	01, 02, ..., 31	(9)
`%b`	Mês como nome da localidade abreviado.	Jan, Feb, ..., Dec (en_US); Jan, Feb, ..., Dez (de_DE)	(1)
`%B`	Mês como nome completo da localidade.	January, February, ..., December (en_US); janeiro, fevereiro, ..., dezembro (pt_BR)	(1)
`%m`	Mês como um número decimal com zeros a esquerda.	01, 02, ..., 12	(9)
`%y`	Ano sem século como um número decimal com zeros a esquerda.	00, 01, ..., 99	(9)
`%Y`	Ano com século como um número decimal.	0001, 0002, ..., 2013, 2014, ..., 9998, 9999	(2)
`%H`	Hora (relógio de 24 horas) como um número decimal com zeros a esquerda.	00, 01, ..., 23	(9)
`%I`	Hora (relógio de 12 horas) como um número decimal com zeros a esquerda.	01, 02, ..., 12	(9)
`%p`	Equivalente da localidade a AM ou PM.	AM, PM (en_US); am, pm (de_DE)	(1), (3)
`%M`	Minutos como um número decimal, com zeros a esquerda.	00, 01, ..., 59	(9)
`%S`	Segundos como um número decimal, com zeros a esquerda.	00, 01, ..., 59	(4), (9)
`%f`	Microsecond as a decimal number, zero-padded to 6 digits.	000000, 000001, ..., 999999	(5)
`%z`	Diferença UTC no formato `±HHMM[SS[.ffffff]]` (string vazia se o objeto é ingênuo).	(vazio), +0000, -0400, +1030, +063415, -030712.345216	(6)
`%Z`	Nome do fuso horário (string vazia se o objeto é ingênuo).	(vazio), UTC, GMT	(6)
`%j`	Dia do ano como um número decimal, com zeros a esquerda.	001, 002, ..., 366	(9)
`%U`	Week number of the year (Sunday as the first day of the week) as a zero-padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0.	00, 01, ..., 53	(7), (9)
`%W`	Week number of the year (Monday as the first day of the week) as a zero-padded decimal number. All days in a new year preceding the first Monday are considered to be in week 0.	00, 01, ..., 53	(7), (9)
`%c`	Representação de data e hora apropriada da localidade.	Tue Aug 16 21:30:00 1988 (en_US); Di 16 Aug 21:30:00 1988 (de_DE)	(1)
`%x`	Representação de data apropriada de localidade.	08/16/88 (None); 08/16/1988 (en_US); 16.08.1988 (de_DE)	(1)
`%X`	Representação de hora apropriada da localidade.	21:30:00 (en_US); 21:30:00 (de_DE)	(1)
`%%`	Um caractere literal `'%'`.	%	

In [ ]:
```
print("O dia da semana da data ", dt.strftime("%d/%m/%Y"), " é ", dt.strftime("%a"))
```

Também é possível converter uma string em um **datetime** com o método **strptime**. O primeiro argumento é a string com a data e o segundo o formato em que a data se encontra.

In [ ]:
```
string_tempo1 = "12/06/2022"
dt2 = datetime.strptime(string_tempo1, "%d/%m/%Y")
print(dt2.day, dt2.year)

string_tempo2 = "12-06-2022"
dt3 = datetime.strptime(string_tempo2, "%d-%m-%Y")
print(dt3.day, dt3.year)
```

Podemos realizar operações entre os objetos datetime, sempre gerando um novo objeto. Considere a subtração de duas datas gerando um objeto do tipo **timedelta** (https://www.geeksforgeeks.org/python-datetime-timedelta-function/):

In [ ]:
```
data1    = "10/01/2022 08:20"
data2    = "10/01/2022 10:00"

dt_data1 = datetime.strptime(data1,"%d/%m/%Y %H:%M")
dt_data2 = datetime.strptime(data2,"%d/%m/%Y %H:%M")

delta_dt1_dt2 = dt_data2 - dt_data1
print("O tempo entre as duas datas é de : ", delta_dt1_dt2)
print("Ou ... {0:.2f} horas".format(delta_dt1_dt2.seconds/3600))
```

Booleanos

Os dois valores booleanos em Python são escritos como **True** e **False**. Comparações são avaliadas como **True** ou **False**. Valores booleanos são combinados com as palavras reservadas **and** e **or**

In [ ]:
```
True and True
```

In [ ]:
```
False or True
```

Casting de tipos

Os tipos str, int e float também são funções que podem ser usadas para cast de valores (ou seja, transformar um tipo em outro). Por exemplo, quando lêmos um valor do usuário pelo método **input()** o valor é lido como uma string. Se tentarmos realizar operações numéricas um erro ocorrerá:

In [ ]:
```
s_numero = input("Digite um número:")
print(s_numero + 10)
```

Podemos então ler o valor e fazer o **cast** para um tipo inteiro:

In [ ]:
```
# Transformando a string em inteiro, e só depois realizando uma operação na mesma
s_numero = int(input("Digite um número:"))
print(s_numero + 10)
```

2.7 Controle de fluxo (if, elif e else)

A instrução **if** é um dos tipos de controle mais conhecidos. Ele verifica uma condição que, se for **True**, fará o bloco que vem a seguir ser avaliado.

OBSERVAÇÃO IMPORTANTE: Bloco de instrução é o conjunto das instruções que estejam num mesmo nível de indentação, mesmo nível hierárquico. No Pyhton os blocos são diferenciados APENAS por dois pontos (:) e pela indentação, ou seja, pelos "recuos" nas linhas. Diferetemente de C++, R ou mesmo JavasCript, que os blocos são delimitados por chaves!

In [ ]:
```
x = 10
if x < 15:
    print("X é maior do que 15")
```

Podemos ainda encadear um if em seguida de outro:

In [ ]:
```
x = int(input("Digite um número inteiro :"))
if  x < 10:
    print("O número é < 10")
elif x < 20:
    print("O número está entre [10,19]")
else:
    print("O número é >= 20")
```

ifs aninhados devem manter a hierarquia da indentações:

In [ ]:
```
a = 21
b = 20
c = True

if (a + b) > 30:
    if a > b:
        print("a + b são maiores do que 30, e a > b")
    else:
        print("a + b são maiores do que 30, e a < b")
else:
    if a > b:
        print("a + b são menores do que 30, e a > b")
    else:
        print("a + b são menores do que 30, e a < b")
```

2.8 Laços de repetição

Os laços de repetição executam um bloco de ações até que uma condição seja satisfeita.

Laços for

O laço **for** serve para iterar em uma coleção (como uma lista ou uma tupla)

In [ ]:
```
lista = [1,2,"string",True,0.85]

# Usando for para iterar sobre os elementos da lista
for e in lista:
    print(e)
```

Podemos parar a execução de um laço for com a palavra **break**

In [ ]:
```
# Para a impressão dos elementos da lista caso algum elemento seja == 0.85
for e in lista:
    if e == 0.85:
        break
    else:
        print(e)
```

Podemos iterar a lista pelos indices. Usamos a função **range()**. Esta função cria um iterador que produz uma sequência de inteiros uniformemente espaçados. Podemos passar o inicio e o fim-1 dos números gerados, ou somente o fim-1, e o range gera a partir do 0.

In [ ]:
```
for i in range(0,10):
    print(i)
```

In [ ]:
```
for i in range(3,10):
    print(i)
```

Usando indices negativos criamos um range reverso, para isso precisamos passar como argumentos o inicio,o fim e o passo (negativo). No exemplo abaixo o inicio é 10 até 0, de -1 em -1.

In [8]:
```
for i in range(10,-1,-1):
    print(i)
```

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

Dessa forma, usamos a função **len()** que determina o tamanho de um objeto, junto a função range para iterarmos em uma lista:

In [ ]:
```
print("A lista contém {} elementos, são eles: ".format(len(lista)))
for i in range(len(lista)):
    print(lista[i])
```

Laços while

Um laço **while** especifica uma condição e um bloco de código que deverá ser executado até a condição ser avaliada como False, ou o laço seja explicitamente encerrado com o **break**.

In [ ]:
```
num = 10
while num < 30:
    num = num + 3
    print(num)
```

O laço abaixo pede um input do usuário até que a letra "a" seja digitada, caso contrário o laço não para:

In [ ]:
```
parada = ""
while parada != "a":
    parada = input("Digite a letra mágica, que eu paro de aparecer...:")
```

Exercícios

1. Crie uma lista com os elementos [10,20,30,1,-1,50,800,500,-600] e imprima as seguintes informações, formatadas em uma frase. DICA: Procure os métodos das listas. OBS: As operações devem ser feitas automaticamente na lista, ou seja, se uma nova lista for passada as operações devem continuar válidas.
   A. Número de elementos da lista
   B. Maior e menor elementos da lista
   C. Imprima a lista em ordem crescente
   D. Imprima a lista em ordem decrescente
2. Faça um programa que leia 2 strings e informe o conteúdo delas seguido do seu comprimento. Informe também se as duas strings possuem o mesmo comprimento e são iguais ou diferentes no conteúdo.
3. Faça um programa que solicite o nome do usuário e imprima-o na vertical
4. Modifique o programa anterior de forma a mostrar o nome em formato de escada.
5. Escreva um programa que receba um aniversário como entrada e exiba a idade do usuário e o número de dias até o seu próximo aniversário
6. Considerando a seguinte sentença atribuida a Sêneca: "Se vives de acordo com as leis da natureza, nunca serás pobre; se vives de acordo com as opiniões alheias, nunca serás rico." Salve a sentença em uma variável e realize as seguintes operações:
   A. Receba uma palavra do usuário e imprima quantas vezes a palavra aparece na frase.
   B. Receba duas palavras do usuário (p1 e p2) e imprima a frase substituindo todas as vezes que p1 ocorre por p2.
7. Escreva um código que leia duas datas de nascimento de duas pessoas fornecidas pelo usuário e realize as seguintes operações:
   A. Imprima quantos anos de vida cada pessoa têm.
   B. Imprima qual é a pessoa mais velha.
   C. Imprima a diferença, em anos, das duas pessoas.
8. Um mercado vende 2 tipos de maças, M1 e M2, se o cliente comprar menos de 6 maças do tipo M1 o custo unitário/maça é de R$ 0.50, caso compre 6 ou mais o custo unitário passa a ser de R$ 0.30. A mesma coisa acontece com as maças do tipo M2, porém abaixo de 10 unidades o preço é de R$ 0.40 e a partir de 10 passa a ser de R$ 0.20. OBS: Considere que uma pessoa só possa comprar maças de um dos dois tipos. Escreva um algoritmo que leia o tipo de maça e a quantidade a ser comprada pelo usuário, e determine o custo total da compra
9. Escreva um algoritmo que realize a soma do 400 primeiros números inteiros pares (incluindo 400).
10. Considere a seguinte lista com as notas dos alunos do curso de Introdução à Mineração de Dados: N = " [0.5,1.8,10,5.9,7.8,9.5,10,2.5,3,6.9,8,2.5,5.9,10,9.8,7]. Salve as notas em uma lista e calcule as seguintes estatísticas:
    A. Maior nota.
    B. Menor nota.
    C. Média da turma.
    D. Para que a turma seja considerada 'boa', a média deve ser > 7. Imprima se a turma é 'boa' ou não.
    E. Calcule o número de alunos que tiraram notas acima e abaixo da média (7).
11. Escreva um algoritmo que leia números informados pelo usuário, e a cada número lido imprima a soma de todos os números digitados até momento, bem como o maior valor. O algoritmo deve parar a sua execução quando o usuário digitar um número negativo.