SPRINT 1

Cap1 - Comando básicos e variáveis

---

os comentários podem ser feito com uma ou mais linhas usando comandos # ou ’’’

```
#Comentário_de_uma_linha
’’’Comentário com
mais de uma linha’’’
```

---

as variáveis podem ser do tipo int, float, etc… Para declarar uma variável, basta escrever seu nome e atribuir algum dado.

Ex:

```
>>>var1 = 10
>>> var2, var3, var4 = 11, 12, 13
```

São variáveis que derivam das datas do tipo inteiro (True ou False).

Na tela de comando, as comparações são interpretadas como perguntas e retornam True ou False.

Ex:

```
>>>14<15
        True
        >>>29==10
        False
```

```
>>>x = int(input(“Digite um número:”))
        5   #Entrada de dado
        >>>x     #Mostra o valor da variável x
        5
```

---

1)

*-> operador

'hello'-> valor

-88.8-> valor

- -> operador

/ -> operador

+ -> operador

5 -> valor

2)

Spam -> Variável

'spam' -> string

3)
string, inteiro, float

4)
uma expressão é constituída por valores e operadores,retornando um único valor.

5)
uma expressão retorna um valor, uma declaração não.

6)
20

7)
spamspamspam

8)
variáveis precisam começar com letras

9)
int(), float(), str()

10)
apresenta o erro: "can't concatenate different type of value mix", use 'str(99) para corrigir o erro.

Cap2 - Condições, laços e módulos

O comando if e else pode ser usado para condição de comparação simples ou múltipla.
Ex:

```
>>>if var1 == True:
                print(“texto”)
        else:
                print(“texto2)
```

Nas comparações múltiplas o python da preferência para True, se uma condição das duas ou mais for False, a comparação final vai ser False
Ex:

```
>>>1 < 3 < 2
```

A primeira comparação é True, mas a Segunda é False, logo a comparação final vai ser False.

Elif: após não passar em uma condição, pode-se usar o elif para testar outra condição.

---

o laço while é usado para repetir um comando até chegar em uma condição.
Ex:

```
base = int(input("Digite a base:"))
ex = int(input("Digite o expoente:"))
cont= 0
prod = 1
while cont<ex:
                    prod = prod*base
                    cont = cont+1
print(base,"^",ex,"=",prod)
```

é um laço while dentro de outro while.
Ex:

```
      i=0
n=5
while i<n:
                    j=0
   while j<n:
                              print(i,j)
                              j=j+1

          i=i+1
```

O for é um laço e pode ser usado com o Range para criar uma sequência de números de 0 até um número n, por padrão, a sequência é incrementada de 1 em 1, mas pode ser alterado.

for variável in range (valor inicial, valor final, nº de incremento)

Ex:

```
      n=int(input("Digite o valor de n:"))
fatorial=1
for fator in range (n, 1, -1):
                     fatorial *=fator
print ("%i! é igual a %i"%( n, fatorial))
```

São repetições encaixadas.

for variável1 in range (vi, vf, in)
    for variável2 in range (vi, vf, in)

Ex:

```
      alunos=10
medias=[ ]
for i in range (1,alunos+1):
                    notas=0
                              for j in range (1,5):
                              notas+=float(input("Digite a nota %i de 4 do aluno %i de
%i:"%(j,i,alunos)))

          notas /=4
```

```
        medias.append(notas)
num=0
for media in medias:
                    if media>=7.0:
                    num+=1

print("O numero de alunos com media maior do que 7 é",num)
```

```
If condição1 and condição2:
        If condicão1 or condição2:
Not #Negação/contrário
Not True -> retorna False
Not False -> retorna True
```

---

1)
True e False, utilizando T e F maiúsculas, e o restante das palavras com letras minúsculas.

2)
and, or e not

3)
True e True é Verdadeiro
True e False é Falso
F e T é Falso
F e F é Falso
T ou T é Verdadeiro
T ou F é Verdadeiro
F ou T é Verdadeiro
F ou F é Falso
not True é Falso
not False é Verdadeiro

4)
False
F
True
F
F
T

5)
==, !=, <, >, <=, e >=.

6)
== compara dois valores e avalia para um Boolean, enquanto = atribui um valor a uma variável.

7)
uma condição é uma expressão usada para o controle de fluxo do código, ela é avaliada a um valor Boolean

8)
O if contém os três blocos e as linhas print('bacon') e print('ham')

9)
o código:
```
if spam == 1:
   print('Hello')
elif spam == 2:
   print('Howdy')
else:
   print('Greetings!')
```

10)
CTRL + C interrompe um programa preso num loop infinito

11)
break move a execução para fora e logo após um loop
continue move a execução para o começo do loop

12)
todos eles fazem mesma coisa, range(10) percorre de 0 até 10(sem incluir o 10), range(0, 10) diz explicitamente que o loop começa em 0 e range(0, 10, 1) diz explicitamente para incrementar em 1 a variável a cada iteração.

13)
o código:
```
for i in range(1,11):
    print(i)
```

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

14)
esta função pode ser chamada de spam.bacon()

Cap3 - Listas

Uma lista contém nenhum, um ou vários elementos/outras listas dentro dela.
Lista = [a, b, c, d] ou Lista = [ ] ou Lista = [a, b, [1, 2, 3], d]
O primeiro elemento tem o índice 0, o segundo tem índice 1 e assim por diante…
Adicionando elementos usando a função .append()

```
Lista = [1, 2, 3, 4]
Lista.append(5)    #O elemento vai ser adicionado no final da lista
Lista = [1, 2, 3, 4, 5]
```

Substituindo elementos: Lista[indice] = elemento

---

len(Lista) -> função conta a quantidade de elementos de uma lista
Lista.count(variável) -> função conta quantas vezes a variável aparece na lista
Ex: lista=[ ]

```
num=int(input("Digite um numero da sequencia:"))

while num!=-1:
        lista.append(num)
        num=int(input("Digite um numero da sequencia:"))

elemento=int(input("Digite o elemento a ser procurado:"))
cont=0

print("O elemento %i aparece %i vezes na lista"%(elemento,lista.count(elemento)))
```

Usando o método Slice
Ex: L1 = [1, 2, 3]

```
L2 = [ ]
L2 = L1[:]    #L2 vai receber os elementos da L1. L2 é uma lista independente de L1
L2 = [1, 2, 3]
```

Usando o método reverse
Ex: L1 = [1, 2, 3]

```
L2 = [ ]
L2 = L1[::-1]   #L2 vai receber os elementos da L1 ao contrário
```

L2 = [3, 2, 1]
Usando o método remove
Ex: L1 = [1, 2, 3]

```
L1.remove(2)   #nome_da_lista.remove(elemento_a_ser_removido)
```

L1 = [1, 3]

Pop: Lista.pop(indice) #remove elemento da lista
Index: Lista.index(elemento) #mostra o indice do elemento
Insert: Lista.insert(posição,elemento) #adiciona um elemento na lista em uma posição escolhida
Sort: Lista.sort() #organiza os elementos da lista em ordem crescente
Clear: Lista.clear() #apaga os elementos da lista
Copy: Lista.copy() #copia os elementos da lista para outra lista independente

---

O Python compara o primeiro elemento entre duas listas, se forem iguais a comparação vai para os próximos elementos, se ainda forem iguais, compara os próximos elementos e assim por diante…. Se na comparação os elementos não forem iguais, o Python testa a condição e responde True ou False.

---

1)
funções reduzem a duplicação de códigos, fazem os programas menores, fáceis de ler e atualizar.

2)
o código de uma função é executado com ela é chamada, não quando é definida.

3)
def cria uma função.

4)
uma função consiste em uma declaração def e um código interno.
a execução do programa faz uma chamada de função, essa chamada de função avalia a função e retorna um valor.

5)
Existe um escopo global, e um escopo local é criado sempre que acontece um chamada de função.

6)
quando uma função retorna um valor, o escopo local é destruído e suas variáveis locais são esquecidas.

7)
uma chamada de função retorna um valor e pode ser usada, diretamente, como parte de uma expressão.

8)
Se a função não apresenta return, ela retorna None.

9)
uma declaração global força uma variável numa função se referir a variável global

10)
None é um tipo de informação NoneType

11)
import importará o módulo chamado areallyourpetsnamederic (não é um módulo oficial do Python)

12)
está função pode ser chamada por spam.bacon()

13)
use try na linha do código que pode causar um erro

14)
try para códigos que podem causar erros
except para executar um código caso ocorra um erro

Cap4 – Funções

Uma função contém um trecho de código que pode ser executado toda vez que for chamada pelo seu nome definido e passando os argumentos que ela precisa para funcionar.
Elementos:
Def: declara a construção da função
Nome_da_função
Argumentos
Valores/variáveis de retorno
Return: indica o fim da função e os valores a serem retornados

```
def nome_da_função (arg1, arg2, …,argn)
        Script
        Return val1, val2,...,valn
```

As tuplas são imutáveis e são definidas com parênteses ou sem.

```
        T = (1, 2, 4) ou T = 1, 2, 4
```

Ex:

```
        T = 1, 2.3, 4.8
        a, b, c, = t   #nas variáveis a, b, c, estão sendo atribuídas os valores da tupla
        a = 1  b = 2.3  c = 4.8
```

É possível colocar listas dentro de tuplas: (1, 2, 3, [4, 5, 70], 9, 6)

Imprimindo elementos de uma tupla com o laço for:

```
        For i in t:
                print(t)
```

Def nome_da_função(*arg)
Ao colocar um asterisco antes de um argumento, o usuário está dizendo para o python que o número de argumentos que estão sendo passado para a função não é conhecido, então o python deve pegar todos os argumentos e compactar em uma tupla.
Ex:

```
def soma(*Lista):   #um argumento declarado
        soma = 0
        for num in Lista:
                soma += num
        return soma
print(soma(1,2,3,4))   #quatro argumentos estão sendo passados
```

Argumentos predefinido: def nome_da_função(arg1, arg2 = valor)

nome_da_função(3)
#Se não for informado o segundo valor na chamada da função, o arg2 vai assumir o valor predefinido.

OBS: argumentos predefinidos devem ser colocados no final.
Ex:

```
def soma(*nums):
        soma = 0
         for num in nums:
                soma += num
         return soma


def media(P1,P2, P3, PESO1 =1 , PESO2 =1, PESO3=2):
         return(P1*PESO1 + P2*PESO2 + P3*PESO3)/soma(PESO1, PESO2, PESO3)

print(media(5, 5, 5))
```

Uma variável local só existe/está definida dentro da função e a variável global está fora da função. Para alterar uma variável global com uma função sem usar o return, deve-se usar o statement global nome_da_variável.
Ex:

```
X=10
def incrementa()
        global X
        incrementa = 5
        X += incrementa
incrementa()
print(X)
```

OBS: a variável global deve estar definida antes da chamada da função.

É uma função que referencia ela mesma
Ex:

```
def fatorial (n)
    If n==1
        return n
    return fatorial(n-1)*n
print(fatorial(5))
```

---

1)
The empty list value, which is a list value that contains no items. This is similar to how '' is the empty string value.

2)
spam[2] = 'hello' (Notice that the third value in a list is at index 2 because the first index is 0.)

3)
'd' (Note that '3' * 2 is the string '33', which is passed to int() before being divided by 11. This eventually evaluates to 3. Expressions can be used wherever values are used.)

4)
'd' (Negative indexes count from the end.)

5)
['a', 'b']

6)
1

7)
[3.14, 'cat', 11, 'cat', True, 99]

8)
[3.14, 11, 'cat', True]

9)
The operator for list concatenation is +, while the operator for replica- tion is *. (This is the same as for strings.)

10)
While append() will add values only to the end of a list, insert() can add them anywhere in the list.

11)
The del statement and the remove() list method are two ways to remove values from a list.

12)
Both lists and strings can be passed to len(), have indexes and slices, be used in for loops, be concatenated or replicated, and be used with the in and not in operators.

13)
Lists are mutable; they can have values added, removed, or changed. Tuples are immutable; they cannot be changed at all. Also, tuples are written using parentheses, (and), while lists use the square brackets, [and].

14)
(42,) (The trailing comma is mandatory.)

15)
The tuple() and list() functions, respectively

16)
They contain references to list values.

17)
The copy.copy() function will do a shallow copy of a list, while the copy.deepcopy() function will do a deep copy of a list. That is, only
copy.deepcopy() will duplicate any lists inside the list.

Cap5 – Dicionários

Um dicionário é uma coleção de valores, assim como uma lista, entretanto, os índices para dicionários podem usar vários tipos de dados e não apenas inteiros.

Índices são chamados de keys (chaves) e uma key com valor associado é chamado de key-value pair.

No código, um dicionário é digitado com {}.

Os itens do dicionário, ao contrário das listas, não são ordenados. No dicionário não importa em qual ordem a key-value pair é digitada.

ex:

```
>>> spam = ['pen', 'note', 'eraser']
>>> materials = ['note', 'pen', 'eraser']
>>> spam == materials
False
>>> body = {'hair': 'hand', 'head': 'foot', 'belly': 'leg'}
```

```
>>> person = {'hand': 'head', 'leg': 'hair', 'foot': 'belly'}
>>> body == person
True
```

Esses três métodos retornam valores de listas mesmo não sendo verdadeiras listas.
Esses tipos de dados podem ser usados em loops.

Exemplo:

```
>>> spam = {'animal': 'cat', 'age': 2}
>>> for v in spam.values():
print(v)
cat
2
```

---

1)
Two curly brackets: {}

2)
{'foo': 42}

3)
The items stored in a dictionary are unordered, while the items in a list are ordered.

4)
You get a KeyError error.

5)
There is no difference. The in operator checks whether a value exists as
a key in the dictionary.

6)
'cat' in spam checks whether there is a 'cat' key in the dictionary, while 'cat' in spam.values()
checks whether there is a value 'cat' for one of the keys in spam.

7)
spam.setdefault('color', 'black')

8)
pprint.pprint()

Cap6 - String

String é uma frase, palavra ou caractere
Concatenando uma string:

```
>>>x = 'Universidade'
y = 'Federal'
z = 'de'
t = 'Uberlândia'
x + y + z + t
'Universidade Federal de Uberlândia'
y = 7 * '-'
x + y
'Universidade-------'
```

OBS: \n enter; \t tab.
Convertendo uma lista/outro tipo de dado em string:

```
>>>str(17)
'17' #caractere 17
```

%i e %d são tipo int
%f e %d são tipo float
%s é string
Ex:

```
nome = input("Digite seu nome:")
Eduardo   #entrada de dado
x = "Olá %s, seja bem-vindo!"%(nome)
x
'Olá Eduardo, seja bem-vindo'
```

Operador end: imprime as strings na mesma linha com um espaço entre elas

```
print("Universidade", end = ' ')
print("Federal", end = ' ')
print("de", end = ' ')
print("Uberlândia")
'Universidade Federal de Uberlândia'
```

Operador end também pode ser usado no laço for

```
for i in range (1,11)
        print(i, end = ' ')
1, ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10
```

As strings são tuplas de caracteres.
"UFU" equivale a "U" + "F" + "U" e a L=['U', 'F', 'U']

```
for char in L
        print char
UFU
```

Função len: conta quantos caracteres tem a tupla

```
palavra = 'programação em python'
len(palavra)
21
```

O operador in, verifica se uma string está em uma tupla.

```
>>>'UFU' in 'UFU Uberlândia'
True
>>>'Ufu' in 'UFU Uberlândia'
False   #caracteres minúsculos se diferem dos maiúsculos
```

O operador .upper() transforma todos os caracteres de uma tupla para maiúsculo e .lower() para minúsculo

```
>>>faculdade = 'Ufu'
>>>faculdade.upper()
'UFU'
>>>faculdade.lower()
'ufu'
```

O operador .find() verifica quantas vezes um caractere aparece em uma palavra

```
>>>'Mississipi'.find('ss')
2
```

O operador .replace(antigo,novo) substitui uma substring por uma outra toda vez que ela aparecer

```
>>>'Mississipi'.raplace('ss','zzz')
'Mizzzizzzii'
```

---

1)
Escape characters represent characters in string values that would otherwise be difficult or impossible to type into code.

2)
\n is a newline; \t is a tab.

3)
The \\ escape character will represent a backslash character.

4)
The single quote in Howl's is fine because you've used double quotes to mark the beginning and end of the string.

5)
Multiline strings allow you to use newlines in strings without the \n escape character.

6)
The expressions evaluate to the following: 'e'
  'Hello'
  'Hello'
  'lo world!

7)
The expressions evaluate to the following: 'HELLO'
  True
  'hello'

8)
The expressions evaluate to the following:
  ['Remember,', 'remember,', 'the', 'fifth', 'of', 'November.']
  'There-can-be-only-one.'

9)
The rjust(), ljust(), and center() string methods, respectively

10)
The lstrip() and rstrip() methods remove whitespace from the left and right ends of a string, respectively.

Cap7 – Pattern Matching With Regular Expressions

```
# Cap7P1
# removendo whitespace no comeco e fim da string

import re

def remspace(string):
    global stringrem
    bspaceRegex = re.compile(r'\s*$')
    fspaceRegex = re.compile(r'^\s*')
    stringrem = bspaceRegex.sub('', string)
    stringrem = fspaceRegex.sub('', stringrem)
    return stringrem

remspace('    here is the string    ')
print(stringrem)
```

```
# Cap7P2
# Password Tester

import re

def passStrengthTest(passWord):
   lowerRegex = re.compile(r'[a-z]')
   upperRegex = re.compile(r'[A-Z]')
   numRegex = re.compile(r'[0-9]')
   molower = lowerRegex.search(passWord)
   moupper = upperRegex.search(passWord)
   monum = numRegex.search(passWord)
   if len(passWord) < 8:
      print('A senha tem menos de 8 caracteres.')
   elif not molower:
      print('A senha precisa de pelo menos um caractere minusculo.')
   elif not moupper:
      print('A senha precisa de pelo menos um caractere maiusculo')
   elif not monum:
      print('A senha precisa de pelo menos um numero.')
   else:
      print('A senha eh forte!')


print('Qual eh a sua senha?')
passW = input()
passStrengthTest(passW)
```

Cap8 – Reading and Writing Files

```
# Cap8P1

import re


with open('madlibs.txt') as madLibFile:
   madLibContents = madLibFile.read()
madLibWords = list(madLibContents.split())

adjectiveRegex = re.compile(r'ADJECTIVE*')
nounRegex = re.compile(r'NOUN*')
adverbRegex = re.compile(r'ADVERB*')
verbRegex = re.compile(r'VERB*')

for i in range(len(madLibWords)):
   if adjectiveRegex.match(madLibWords[i]):
```

```
        print('Enter an adjective', end=': ')
        sub = input()
    elif nounRegex.match(madLibWords[i]):
        print('Enter a noun', end=': ')
        sub = input()
    elif adverbRegex.match(madLibWords[i]):
        print('Enter an adverb', end=': ')
        sub = input()
    elif verbRegex.match(madLibWords[i]):
        print('Enter a verb', end=': ')
        sub = input()
    else:
        continue
    madLibWords.remove(madLibWords[i])
    madLibWords.insert(i, sub)

joinWordsList = ' '
newString = joinWordsList.join(madLibWords)
with open('madlibsNew.txt', 'w') as newMadLibFile:
    newMadLibFile.write(newString)
print(newString)
```

```
# Cap8P2

import os
import re

dirfiles = os.listdir('C:\\gam')

print('What text do you want to search for?')
userReg = str(input())
stringRegex = re.compile(userReg)
fileRegex = re.compile(r'\w+\.txt')

for i in range(len(dirfiles)):
    if fileRegex.search(dirfiles[i]):
        openFile = open('C:\\gam\\' + dirfiles[i])
        readFile = openFile.readlines()
        for line in range(len(readFile)):
            r = 0
            if stringRegex.search(readFile[r]):
                print(readFile[r])
                r = r + 1
```
