MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

1º Semestre - 2023

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

Chapter 6 Iteration (“Iteração”)

6.1 Multiple assignment (“Múltiplas atribuições”)

Considere esse exemplo didático simples:

```
bruce = 5

print(bruce, end=" ")

bruce = 3

print(bruce)
```

A variável *bruce* recebe o valor 5

A variável *bruce* é impressa

A variável *bruce* recebe o valor 3

A variável *bruce* é impressa

```
In [2]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/untitled0.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')
5 3
```

O uso de end=“ “ é recurso para evitar quebra de linha

A mesma variável ao longo do código pode receber diversos valores servindo a algum propósito determinado pelo usuário.

Como promover essa variação, de forma compacta, sem sucessivas atribuições ?

6.2 The while statement ("O comando *while*")

O principal uso da computação é a execução de tarefas repetitivas de forma consistente.

Retomando uma das funções exemplo que usamos anteriormente (com recursão):

```
1  def countdown(n):
2      while (n > 0):
3          print (n)
4          n = n-1
5      print ("Lift-off!")
6
7
8  countdown(10)
9
```

A função tem como entrada o valor n

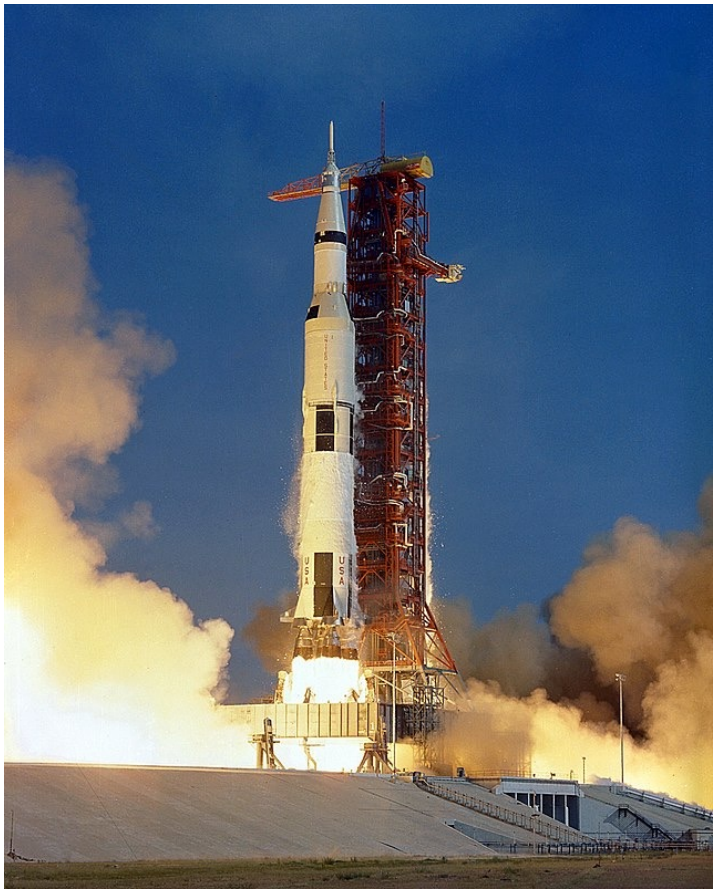
O que estiver no laço do *while* será executado enquanto a condição é TRUE.

Para quebrar o laço é necessária a variação de algum parâmetro que afeta a condição do *while*.

Após o laço ser executado o comando seguinte (no nível do while) é executado.

O resultado é idêntico ao obtido anteriormente.

```
In [7]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/teste_iter2.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
10
9
8
7
6
5
4
3
2
1
Lift-off!
```



No caso do exemplo anterior fica claro que o valor de *n* irá decrescer até que a condição de quebra do loop seja alcançada. Em outros exemplos a variação da condição pode não ser monotônica.

Por exemplo:

A função tem como entrada o valor *n*

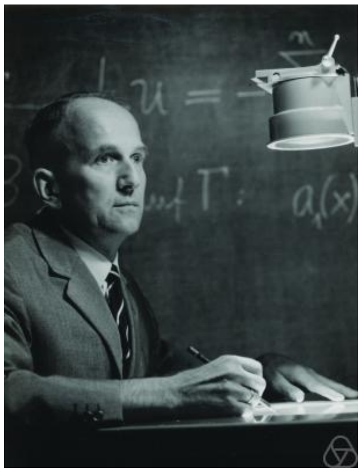
O que estiver no laço do *while* será executado enquanto a condição é TRUE, ou seja enquanto *n* for diferente de 1.

Os ramos do condicional aplicam diferentes variações de *n* dependendo de *n* é par ou ímpar.

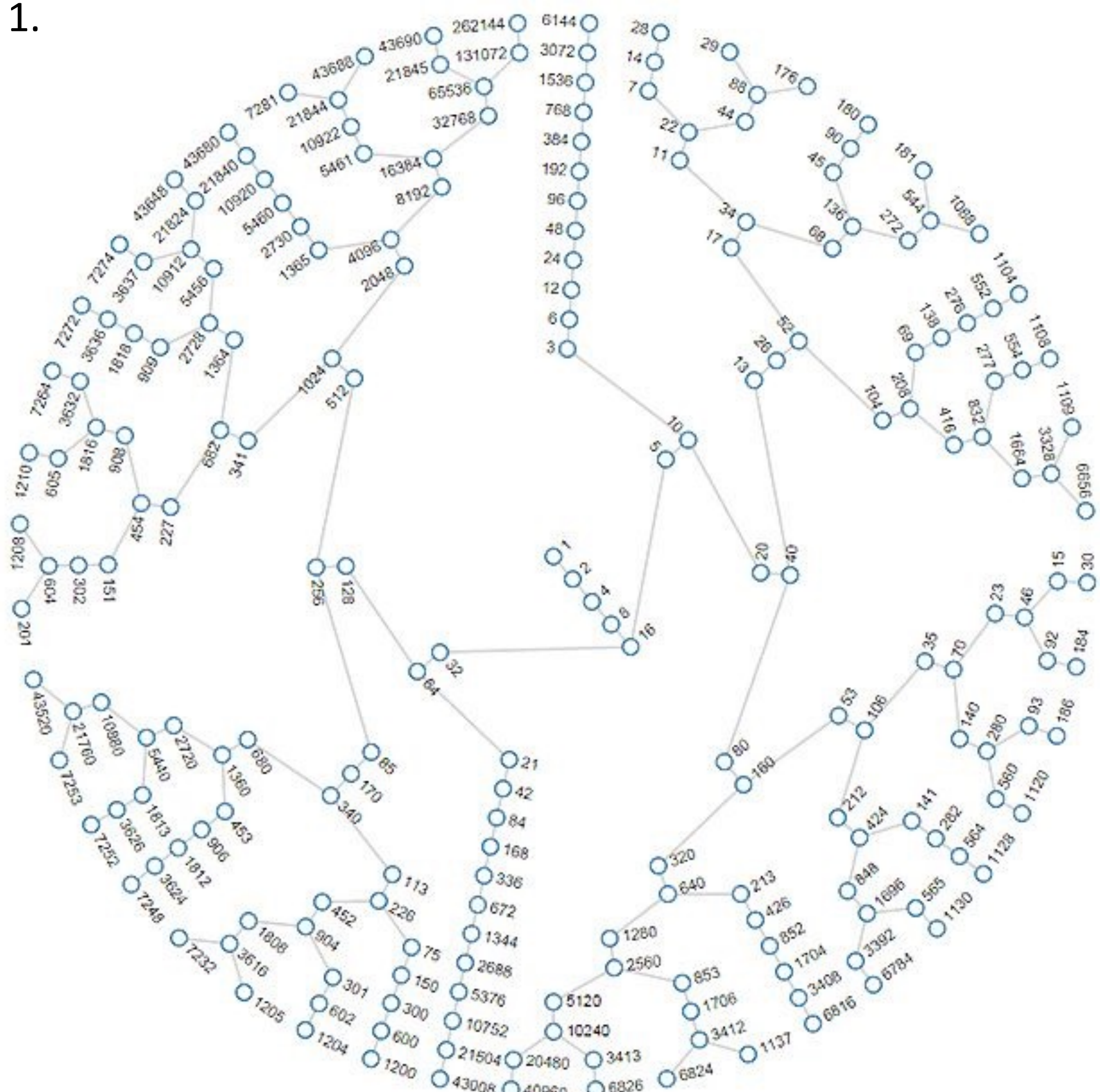
```
1     def sequence(n):
2         while (n != 1):
3             print (n,end=" ")
4             if (n%2 == 0):
5                     n = n //2     # n  par
6             else:
7                     n = n*3+1   # n ímpar
8
9
10    sequence(3)
11
```

```
In [10]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/teste_iter3.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
3 10 5 16 8 4 2
```

Essa função é chamada Conjectura de Collatz que afirma que partindo de qualquer número inteiro > 1 , em algum momento a sequência retorna a 1.



Lothar Collatz
(1910-1990)



As funções apresentadas anteriormente usando recursão podem ser desenvolvidas usando iteração.

```
1   def newLine(n):
2       while (n > 0):
3           print (n, " ")
4           n = n - 1
5
6   print("First Line")
7
8   newLine(3)
9
10  print("Second Line")
```

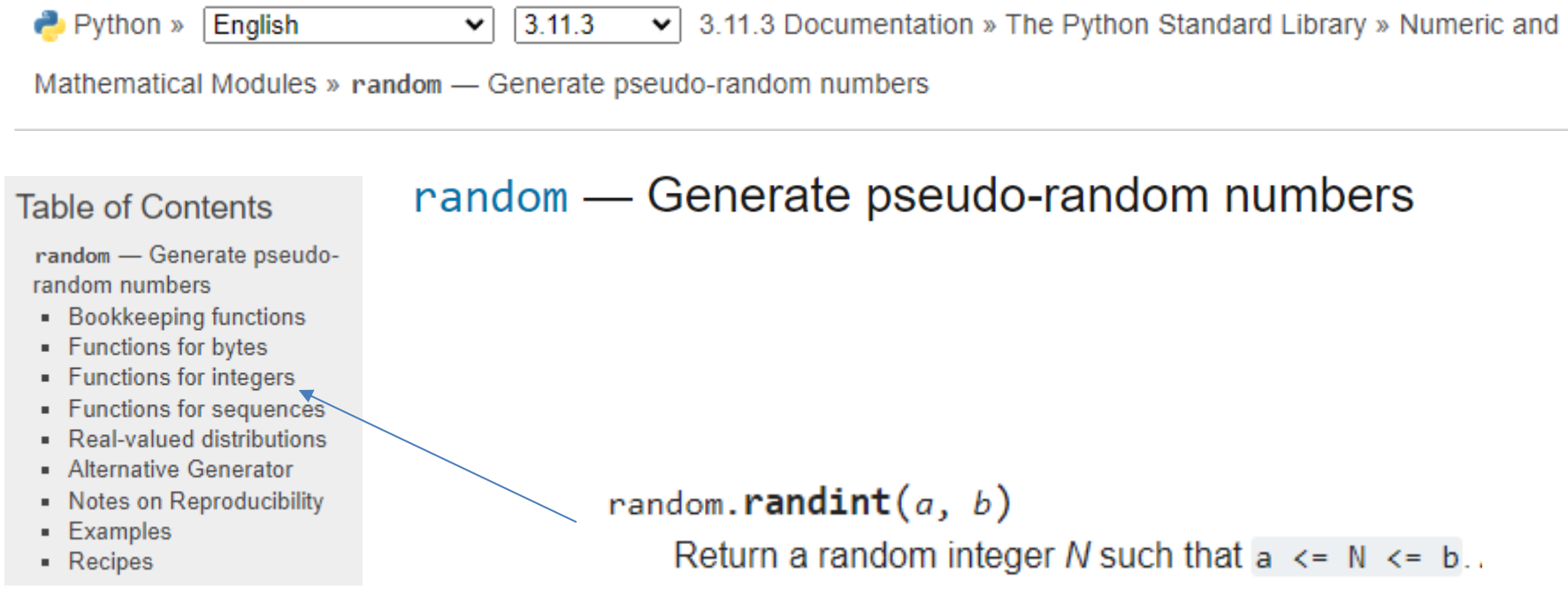
```
In [15]: runfile('C:/Users/win/
remote/MAP2112_2023/scripts/tes
wdir='C:/Users/win/Dropbox/USP/
MAP2112_2023/scripts')
First Line
3
2
1
Second Line
```

```
1   def factorial(n):
2       fact = 1
3       while (n > 0):
4           fact = fact*n
5           n = n - 1
6       return fact
7
8   a = factorial(5)
9   print(a)
```

```
In [13]: runfile('C:/Users/win/
remote/MAP2112_2023/scripts/tes
wdir='C:/Users/win/Dropbox/USP/
MAP2112_2023/scripts')
120
```

Jogo da Adivinhação:

- O usuário escolhe dois valores inteiros.
- O código sorteia um valor entre (e incluindo) os valores limites definidos pelo usuário.
- O usuário “chuta” qual seria esse valor.
- O código diz se o chute está abaixo ou acima do valor sorteado.
- Com essa informação o usuário faz um novo chute.
- Se o usuário acerta o programa para e diz qual é o valor sorteado para conferência.
- O usuário tem 10 tentativas para acertar.



Geradores de números pseudo-aleatórios são ferramentas úteis para simular a imprevisibilidade.

```
1
2   import random
3
4   def dice():
5       return random.randint(1,6)
6
7   print(dice())
8
9   print(dice())
```

A cada execução novos números são sorteados, simulando lançamentos sucessivos de um dado.

```
In [5]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/untitled0.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
6
4

In [6]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/untitled0.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
5
5

In [7]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/untitled0.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
6
5
```

COMO SERIA O CÓDIGO QUE IMPLEMENTA O JOGO ?

- UM EXEMPLO DE IMPLEMENTAÇÃO

```python
import random

print("Jogo da Adivinhação")

n_min = int(input("Entre com o valor mínimo: "))
n_max = int(input("Entre com o valor máximo: "))

tent_max = 10

sorte = random.randint(n_min, n_max)

tent = 0
check = True

while (check):
    chute = int(input("Chute um valor: "))
    tent = tent + 1

    check = (chute != sorte)

    if (chute > sorte):
        print("chutou alto")
    if (chute < sorte):
        print("chutou baixo")

    if (tent >= tent_max):
        check = False

print("Numero de tentativas", tent)

if (tent > tent_max):
    print("Numero máximo de tentativas atingido")

print("O valor sorteado era ", sorte)
```

Entrada com conversão para inteiros dos valores máximo e mínimo

Definição do no. máximo de tentativas

Função de geração de números inteiros aleatórios

Laço principal do código (while)

Captura do chute do usuário

Incremento do no. de tentativas

Verificação do chute

Aviso ao usuário se o chute foi alto ou baixo

Verificação do no. de tentativas

Comunicação do resultado

```
In [3]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_iter6_jogo.py', wdir='C:/
Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
Jogo da Adivinhação
Entre com o valor mínimo: 1
Entre com o valor máximo: 100
Chute um valor: 50
chutou alto
Chute um valor: 25
chutou baixo
Chute um valor: 35
chutou baixo
Chute um valor: 45
chutou alto
Chute um valor: 40
chutou baixo
Chute um valor: 42
chutou baixo
Chute um valor: 43
Numero de tentativas 7
O valor sorteado era  43
```

6.3 Tables

(Tabelas)

Um exemplo didático do uso do laços é a construção de tabelas de funções.

```
1   import math
2
3   x_ini = 1.0
4   x_final = 10.0
5   delta_x = 1.0
6
7   x = x_ini
8   while (x < x_final):
9       print (x, "\t", math.log(x))
10      x = x + delta_x
```

A opção “\t” inclui um TAB

```
In [10]: runfile('C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts/
teste_iter8.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/
scripts')
1.0      0.0
2.0      0.6931471805599453
3.0      1.0986122886681098
4.0      1.3862943611198906
5.0      1.6094379124341003
6.0      1.791759469228055
7.0      1.9459101490553132
8.0      2.0794415416798357
9.0      2.1972245773362196
```

Lembre-se a função math.log(x) é o logaritmo natural (ou neperiano) de base *e* = 2,71828...

Caso a intenção for gerar uma tabela de logaritmo em outra base pode-se usar a relação de mudança de base:

$$\log_b(a) = \frac{\log_c(a)}{\log_c(b)}$$

Alterando o código para base 2 por exemplo:

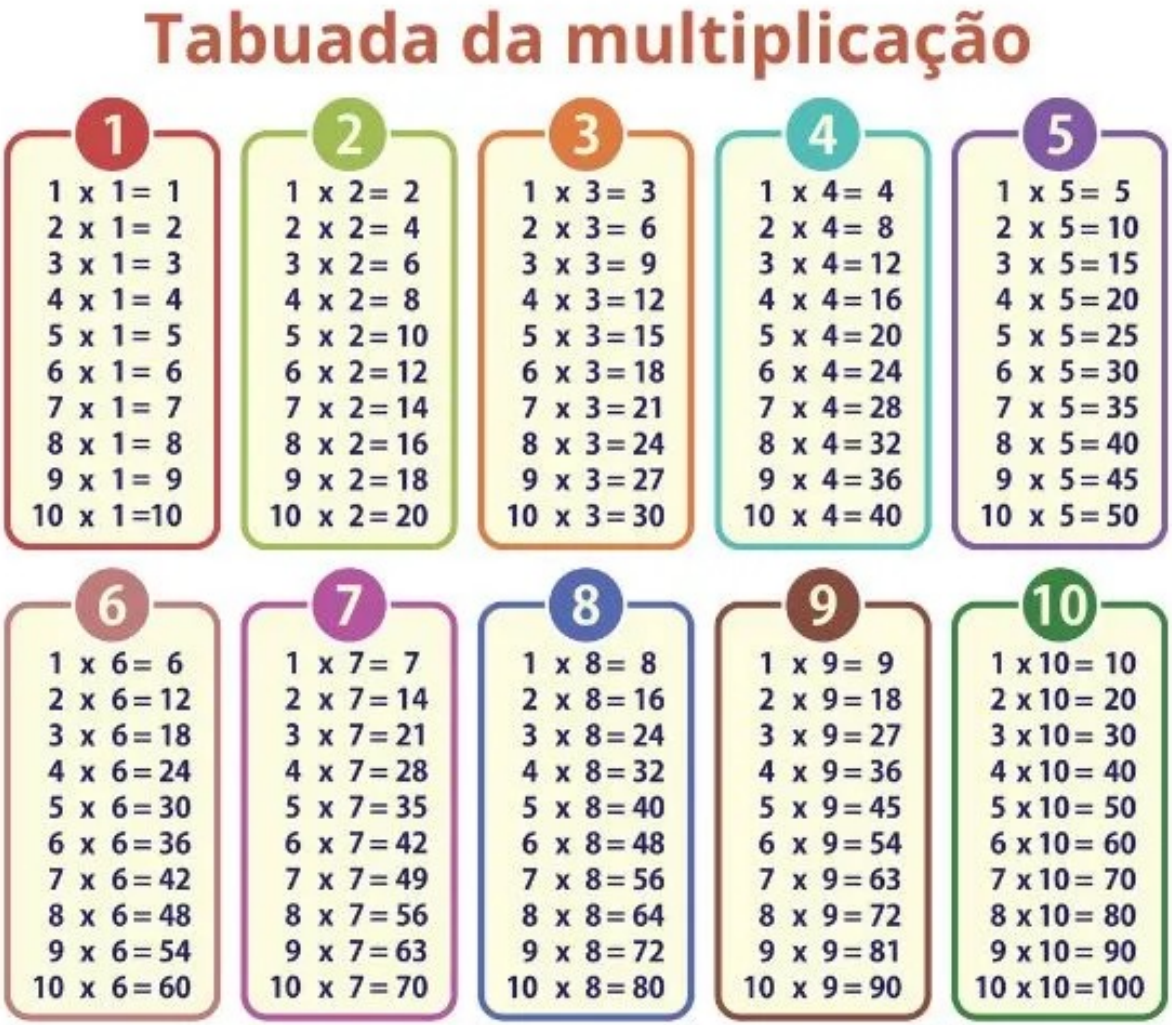
```
1   import math
2
3   x_ini = 1.0
4   x_final = 10.0
5   delta_x = 1.0
6
7   x = x_ini
8   while (x < x_final):
9       print (x, "\t", math.log(x)/math.log(2))
10      x = x + delta_x
```

```
In [11]: runfile('C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts/
teste_iter9.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/
scripts')
1.0      0.0
2.0      1.0
3.0      1.5849625007211563
4.0      2.0
5.0      2.321928094887362
6.0      2.584962500721156
7.0      2.807354922057604
8.0      3.0
9.0      3.1699250014423126
```

6.4 Two-dimensional tables ("Tabelas Bidimensionais")

6.5 Encapsulation and generalization ("Encapsulamento e Generalização")

Tomando como exemplo a tarefa de construir uma tabuada de multiplicação.



A nossa tabela será construída inicialmente com uma função que escreve as linhas

```
1  def printMultiples(n):
2      i = 1
3      while i <= 10:
4          print (n*i,"\t",end=" ")
5          i = i + 1
6
7
8  printMultiples(6)
```

Fazendo para n = 6

```
In [15]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_iter10_tabu.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
6    12      18      24      30      36      42      48      54      60
```

Para completar a tabela basta encapsular a função num novo laço varrendo todas as linhas.

```
1   def printMultiples(n):
2       i = 1
3       while i <= 10:
4           print (n*i," ",end=" ")
5           i = i + 1
6
7   i = 1
8   while (i<=10):
9       printMultiples(i)
10      print()
11      i = i + 1
```

```
In [18]: runfile('C:/Users/win/Dropbox/USP/2023/re
remote/MAP2112_2023/scripts')
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10   12   14   16   18   20
3   6   9   12   15   18   21   24   27   30
4   8   12   16   20   24   28   32   36   40
5   10   15   20   25   30   35   40   45   50
6   12   18   24   30   36   42   48   54   60
7   14   21   28   35   42   49   56   63   70
8   16   24   32   40   48   56   64   72   80
9   18   27   36   45   54   63   72   81   90
10   20   30   40   50   60   70   80   90   100
```

Mais adiante veremos como melhor formatar a saída de dados, além de leitura e gravação de arquivos.

RE-CAPITULANDO O QUE FOI VISTO ATÉ O MOMENTO

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, ...]
dictionary = { key1:value1, key2:value2, ...}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: x,y stand for any kind of data values, s for a string, n for a number, L for a list where i,j are list indexes, D stands for a dictionary and k is a dictionary key.

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Built-in functions

print(x, sep='y')	prints x objects separated by y
input(s)	prints s and waits for an input that will be returned
len(x)	returns the length of x (s, L or D)
min(L)	returns the minimum value in L
max(L)	returns the maximum value in L
sum(L)	returns the sum of the values in L
range(n1,n2,n)	returns a sequence of numbers from n1 to n2 in steps of n
abs(n)	returns the absolute value of n
round(n1,n)	returns the n1 number rounded to n digits
type(x)	returns the type of x (string, float, list, dict ...)
str(x)	converts x to string
list(x)	converts x to a list
int(x)	converts x to a integer number
float(x)	converts x to a float number
help(s)	prints help about x
map(function, L)	Applies function to values in L

Conditional statements

```
if <condition> :
    <code>
else if <condition> :
    <code>
...
else:
    <code>
```

```
if <value> in <list>:
```

Data validation

```
try:
    <code>
except <error>:
    <code>
else:
    <code>
```

Working with files and folders

```
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>)
os.listdir(<path>)
```

Loops

```
while <condition>:
    <code>
```

```
for <variable> in <list>:
    <code>
```

```
for <variable> in
range(start,stop,step):
    <code>
```

```
for key, value in
dict.items():
    <code>
```

Loop control statements

break	finishes loop execution
continue	jumps to next iteration
pass	does nothing

Running external programs

```
import os
os.system(<command>)
```

Functions

```
def function(<params>):
    <code>
    return <data>
```

Modules

```
import module
module.function()
```

```
from module import *
function()
```

Reading and writing files

```
f = open(<path>,'r')
f.read(<size>)
f.readline(<size>)
f.close()
```

```
f = open(<path>,'r')
for line in f:
    <code>
f.close()
```

```
f = open(<path>,'w')
f.write(<str>)
f.close()
```

Legend: x,y stand for any kind of data values, s for a string, n for a number, L for a list where i,j are list indexes, D stands for a dictionary and k is a dictionary key.

Fim Aula 05