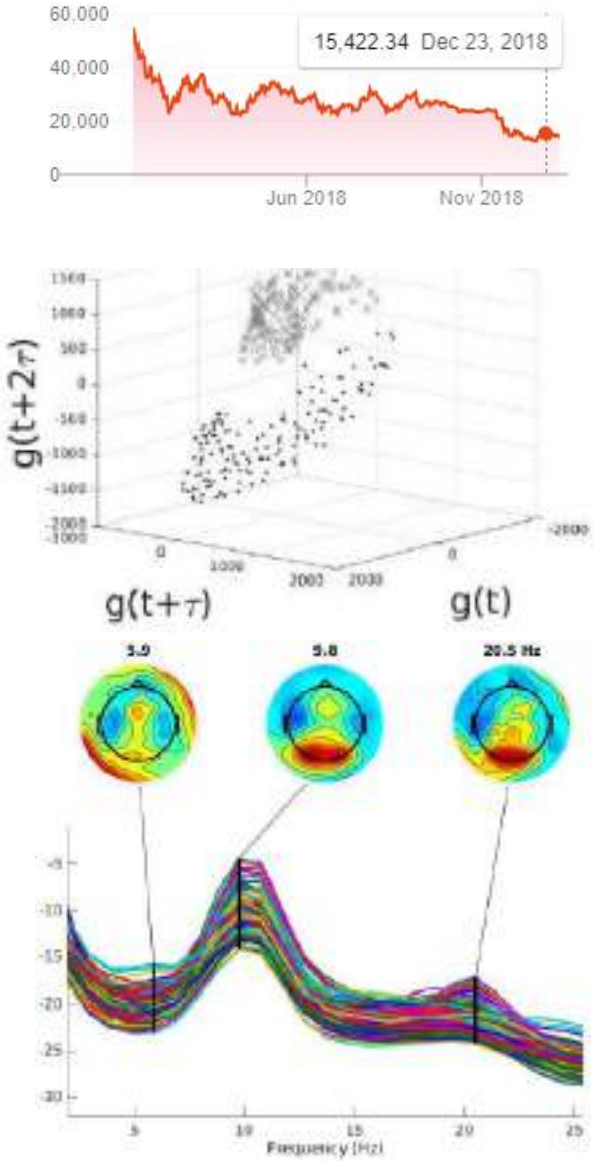


UNICAMP

Instituto de Computação

MC102: Algoritmos e Programação de Computadores (turmas 4,5,6,7)

2 – Variáveis e seus tipos
Operações aritméticas, relacionais e lógicas

ARTHUR VALENCIO

Pós-doutorando IC/Unicamp

Fapesp CEPID NeuroMat

Campinas, 9 de Março de 2019

>>Regras da disciplina: atualização

Como o Moodle encontra-se indisponível, **não teremos as atividades conceituais**

Caso frequência <75%: **Reprovado por frequência**

Caso $M_{provas} < 2,5$ ou $M_{lab} < 2,5$: **Reprovado por nota**

Caso frequência≥75% e $M_{provas} \geq 5$ e $M_{lab} \geq 5$

Aprovado com

$$M_{final} = \max(5; 0,7 \cdot M_{provas} + 0,3 \cdot M_{lab})$$

Caso contrário

EXAME

$$M_{preliminar} = \min(4,9; 0,7 \cdot M_{provas} + 0,3 \cdot M_{lab})$$

$$M_{final} = (M_{preliminar} + M_{exame})/2$$

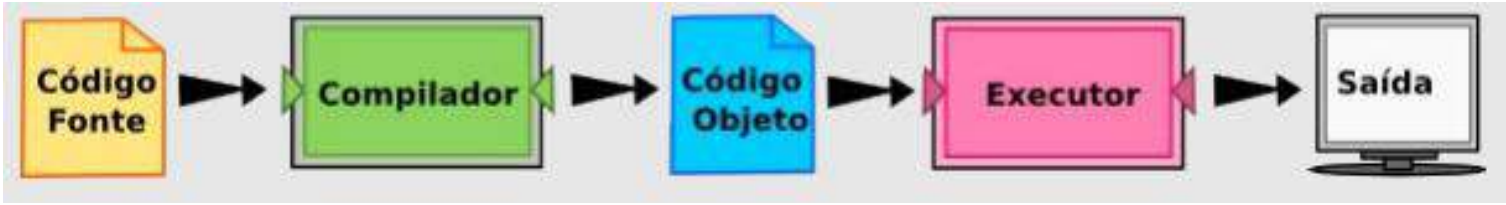
Aprovado se $M_{final} \geq 5$

Reprovado por nota se $M_{final} < 5$

>>Regras da disciplina: atualização

>>Compilador vs interpretador

- **Python** é uma linguagem de **alto** nível, mas máquinas só conseguem processor instruções em **baixíssimo** nível: precisamos de um *tradutor*



Python utiliza os dois processos, mas é principalmente uma linguagem **interpretada**

>>Modo shell vs script

- O interpretador **Python** pode ser utilizado de duas formas:

1 – Você deixa uma sessão aberta e vai digitando comandos e o interpretador Python vai te mostrando o resultado comando-a-comando (modo **shell**/ambiente interativo/console):

Basta abrir o *terminal* ou *prompt de comando* e digitar **python3** ou **python**
O símbolo **>>>** indica que você está rodando Python
Digite quit() para sair

2 – Você escreve o código inteiro, salva como um arquivo e manda o interpretador executar (modo **script**):

Por exemplo, salve o arquivo como *myfile.py*
Abra o terminal/prompt de comando e, usando **cd** mude para a mesma pasta do arquivo
Digite **python3** *myfile.py*
(ou **python** *myfile.py*)

>>Exercício interativo

- Vamos imprimir **“Hello World!”** de várias formas

O que você quiser

Modifique o separador utilizando sep=“_”

O que você quiser

Modifique o comando de fim utilizando end=“_”

Adicione um comentário utilizando **#**

>>Comentando o seu script

- Comentar o seu script é uma boa prática para:
  1 – estabelecer direitos de autoria
  2 – relembrar o que você fez
  3 – facilitar futuras manutenções e correções
  4 – auxiliar os professors a entender a sua linha de raciocínio

Faça isso com o #

Recomendado:
Iniciar o código com informação sobre:
1- descrição do que o código faz
2- o seu nome
3- a data de construção/edição
4- licença de distribuição (C), (CC), domínio público, Apache, etc, se aplicável

Nas etapas mais importantes de cálculo informe o que está sendo feito

Deixe comentado possíveis “breakpoints” de verificação de erros

>>Estrutura de programa Python

- O interpretador Python separa os comandos por linha:

```
print(“Hello”)
print (“World!”)
```

Out:
Hello
World!

```
print(“Hello”) print (“World!”)
```

ERRO: dois comandos de uma vez

Podemos contornar isso com **;**

```
print(“Hello”); print (“World!”)
```

>>Objetos e tipos

- Queremos usar os programas para manipular *dados*
- Todo dado em Python é um **objeto** com um **tipo** definido:
    - **int** : número inteiro (2)
    - **float**: números reais (3.14159265)
    - **complex**: números complexos (3.14+9.2i)
    - **str**: sequências de caracteres (strings) – para texto (“Unicamp é d+!”)
    - **bool**: valores booleanos (True ou False)
    - etc
- A função **type** pode ser usada para descobrir o tipo de um dado
- É muito comum erros devido à mistura de números inteiros com fracionários se utilizar bibliotecas especiais. Para garantir que quer que um número possa ser considerado fracionário, defina-o com o ponto flutuante: **10.0** ou **10.**

>>Variáveis

- Frequentemente precisamos armazenar informações na memória para uso rápido e/ou repetido depois
- A essa informação armazenada atribuimos um **nome** fácil

Algoritmo (pseudocódigo)

$x \leftarrow 10$ (número real)

Python

```
x=10.0
```

Definido com 32 ou 64bits de precisão

>>Variáveis: regras para nomes

- **Python** diferencia maiúsculas e minúsculas
- Não se pode começar o nome de uma variável com um número
- Não se pode utilizar os seguintes caracteres no nome **{ ( ) } + - * / \ ; . , ? $**
- Não se pode utilizar qualquer uma das palavras reservadas:

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
ass	raise	return	try	while	with
yield	True	False	None		

>>Operadores aritméticos

Note como Python 3 **tenta** sempre que necessário converter int para float

- **Adição (+):**

```
>>> 2+3.141595926535979
5.141595926535979

>>> a=3.1415
>>> a+a+3.14
9.423
```

- **Subtração (-):**

```
>>> a=100
>>> b=0.1
>>> a-b
99.9
```

- **Multiplicação (*):**

```
>>> b=3*1.23
>>> print(b)
3.69
```

- **Divisão (/):**

```
>>>x=7/2
>>>x
3.5
```

- **Divisão inteira (//):**

```
>>>y=7/2
>>>y
3
```

- **Módulo – resto da divisão inteira (%):**

```
>>>a=13.73
>>>a%7
6.73
```

- **Exponenciação (**):**

```
>>>2**10
1024

>>>9**0.5
3.0
```

>>Desafio

Construa um programa para calcular a distância euclidiana entre dois pontos

>>Operadores aritméticos: ordem

1. Exponenciação
2. Multiplicação e divisão
3. Módulo
4. Adição e divisão

Em caso de mais de uma operação, ordem é da esquerda para a direita

Podemos controlar a ordem usando parênteses **()**:

```
>>>3*3+4
13
>>>3*(3+4)
21
```

>>Operadores relacionais

- São utilizados para **comparações** entre dois valores
- Seu resultado é **True** ou **False**

- **Igualdade: ==**

```
>>>a=2
>>>b=4**0.5
>>>a==b
True

>>> 2==3
False
```

- **Diferença: !=**

```
>>> 2!=3
True
```

- **Maior: >**

```
>>>a=2
>>>b=4**0.5
>>>a>b
False

>>> 3>2
True
```

- **Menor: <**

```
>>> 2<3
True
```

- **Maior ou igual: >=**

```
>>>a=2
>>>b=4**0.5
>>>a>=b
True

>>> 3>=2
True
```

- **Menor ou igual: <=**

```
>>> 2<=3
True
```

>>Operadores lógicos

- Compara duas condições, o que permitirá o programa tomar decisões

- **E:** *and*
  >>>(2<4) and (3!="3")
  True

X and Y	Y=True	Y=False
X=True	True	False
X=False	False	False

- **Ou:** *or*
  >>(2>4) or (3<=9**0.5)
  True

X and Y	Y=True	Y=False
X=True	True	True
X=False	True	False

- **Negação:** *not*
  >>>x=(3<2) #False
  >>>not x
  True

  >>>y=(not x)
  >>>y
  False

>>Pontos flutuante: precisão

- Como existe uma memória finita na forma que um número é salvo no computador, problemas de precisão podem surgir:

```
>>> 1/10
0.1
```

```
>>>0.1+0.2
0.30000000000000004
```

Pior: esses erros são **cumulativos**

MS211 Cálculo Numérico

Pergunta: quais as vantagens e desvantagens de um sistema 32-bits vs 64-bits?

>>String

- O tipo string é usado para **armazenar texto**
- Pode-se definir através de aspas simples

x=**'Olá classe!'**

ou aspas duplas

y=**"Olá professor Tibúrcio!"**

- Podemos concatenar duas strings usando **+**:

```
>>>print(x+" "+y)
Olá classe! Olá professor Tibúrcio!
```

- Podemos replicar strings usando *:

```
>>>print(3*"Penny! ")
Penny! Penny! Penny!
```

Operadores "aritméticos" com strings

Replicação tem precedência

Também podemos usar operadores relacionais e lógicos com strings

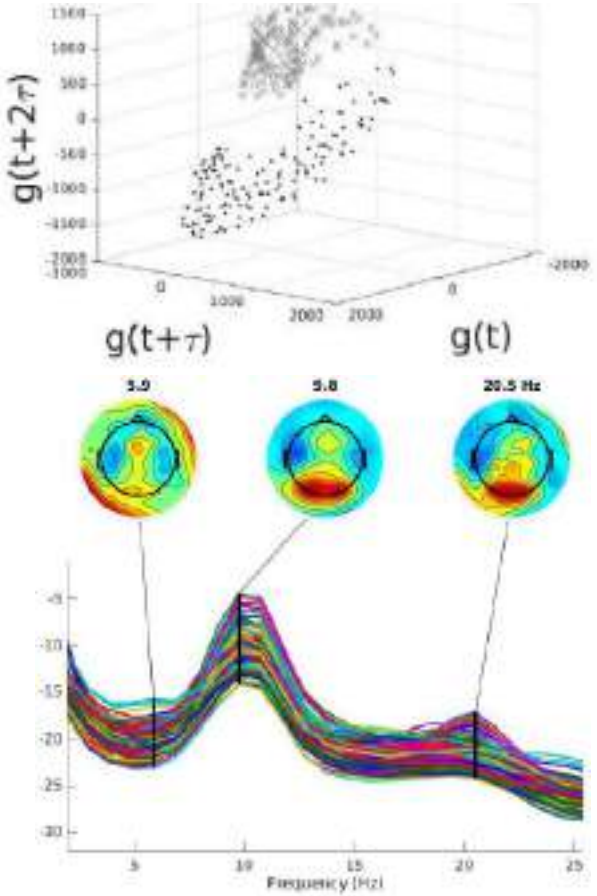
>>Tipagem em Python

- Diferente de outras linguagens (C etc), Python possui **tipagem fraca**
- Isso significa que, a cada momento, podemos atribuir objetos de **diferentes tipos** para uma variável:

```
>>>a=0.2
>>>print(a+2)
2.2
>>>print(type(a))
<class ‘int’>
>>>a=“This is just a flesh wound!”
>>>print(type(a))
<class ‘str’>
```

- Além disso, como vimos, podemos converter entre diferentes tipos usando:
  int(), float(), str()

>>Grupos



UNICAMP

Instituto de Computação

That´s all folks!

arthur_valencio@physics.org

http://www.arthurvalencio.com/mc102

http://www.ic.unicamp.br/~mc102