

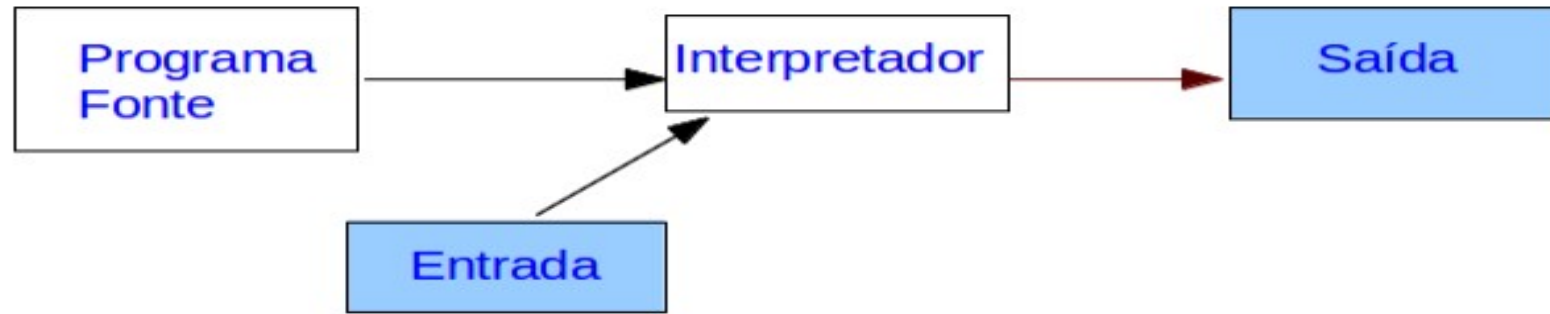


Tópicos Especiais

Introdução

Prof. MSc. Rafael Staiger Bressan
rafael.bressan@unicesumar.edu.br

Python Interpretador



Python

- Muitos recursos
 - Orientação a Objetos
 - Escalável (módulos, classes, controle de exceções)
 - Biblioteca embutida extensa e grande número de módulos fornecidos por terceiros
- Grande variedade de aplicações
- Linguagem interpretada (script)
- Multi-plataforma
- Grátis!
- Comunidade bastante grande

Python

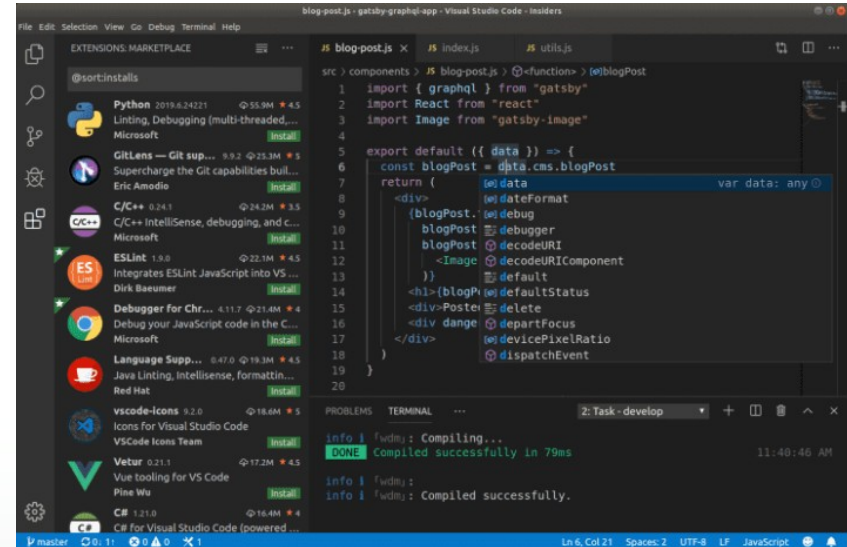
Quem Utiliza?



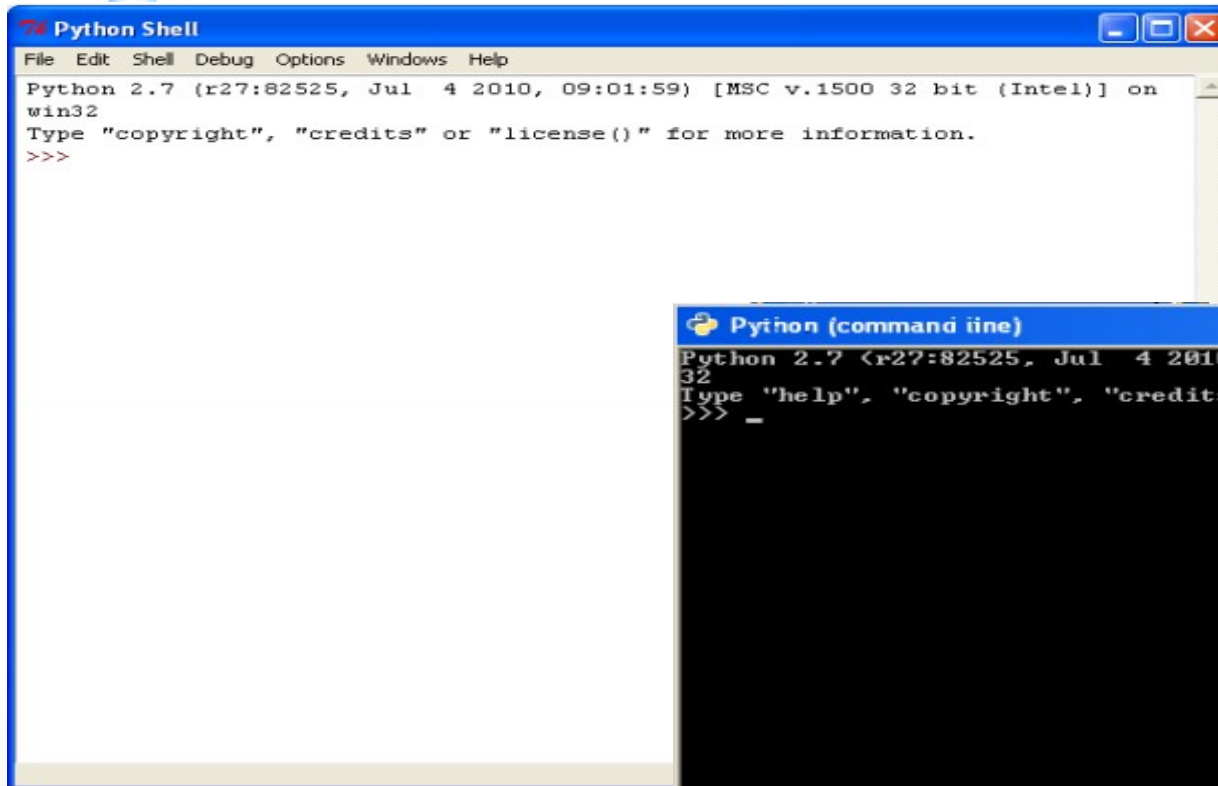
Python

- Instalando o interpretador Python
 - <https://www.python.org>
 - Python 3.x

<https://code.visualstudio.com/>

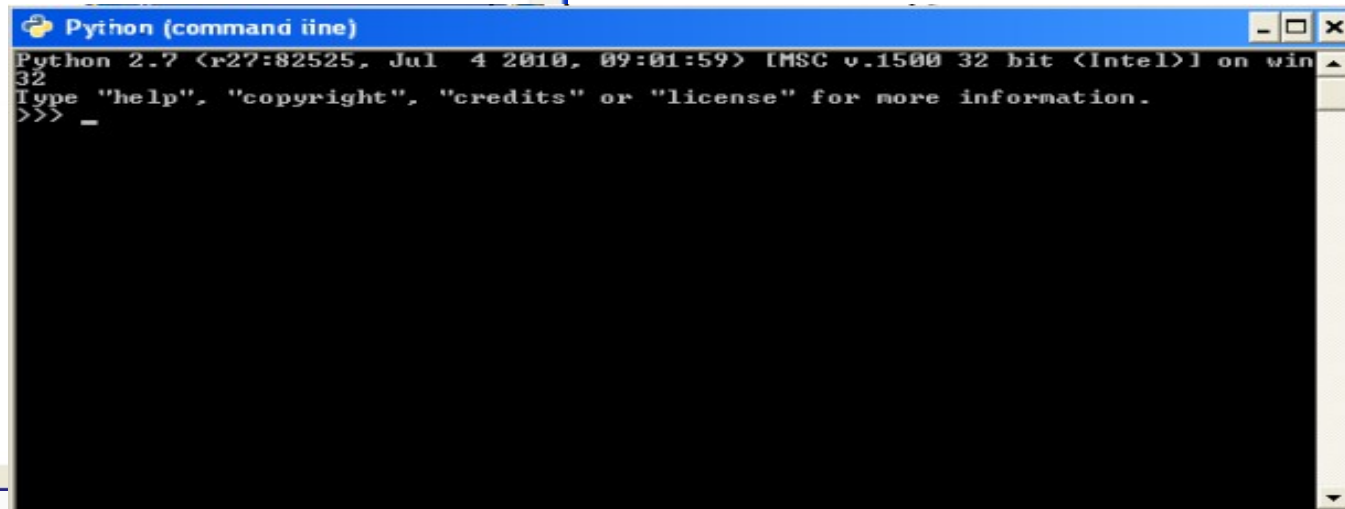


Python



A screenshot of a Windows application window titled "Python Shell". The window has a blue title bar and a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following text: "Python 2.7 (r27:82525, Jul 4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32", "Type \"copyright\", \"credits\" or \"license()\" for more information.", and the prompt ">>>".

```
Python 2.7 (r27:82525, Jul 4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```



A screenshot of a Windows application window titled "Python (command line)". The window has a blue title bar and standard window controls. The main text area shows the following text: "Python 2.7 (r27:82525, Jul 4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32", "Type \"help\", \"copyright\", \"credits\" or \"license\" for more information.", and the prompt ">>> _".

```
Python 2.7 (r27:82525, Jul 4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Python

- Pythontutor - <http://pythontutor.com/>

Start using Online Python Tutor now

For instance, here is a visualization showing a program that recursively finds the sum of a (cons-style) linked list. Click the “Forward” button to see what happens as the computer executes each line of code.

```
1 def listSum(numbers):
2     if not numbers:
3         return 0
4     else:
5         (f, rest) = numbers
6         return f + listSum(rest)
7
8 → myList = (1, (2, (3, None)))
9 → total = listSum(myList)
```

[Edit code](#)



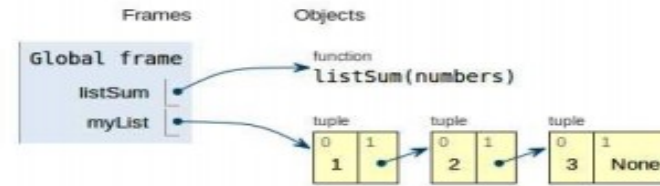
< Back

Step 3 of 22

Forward >

→ line that has just executed

→ next line to execute

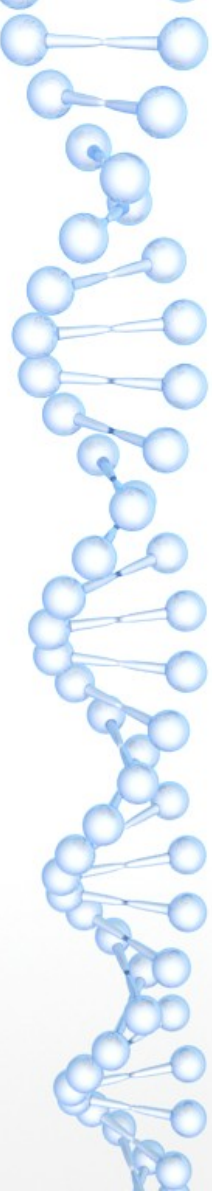




Python como calculadora

- O Interpretador python pode ser usado como calculadora
- Por exemplo, as quatro operações aritméticas são denotadas pelos símbolos
 - + adição
 - subtração
 - * multiplicação
 - / divisão
 - % resto
 - ** potenciação

Python como calculadora



```
>>> 10
10
>>> # Um comentário é precedido do caracter "#"
... # Comentários são ignorados pelo interpretador
... 10+5
15
>>> 10-15 # Comentários podem aparecer também após código
-5
>>> 10*3
30
>>> 10/3 # Divisão inteira retorna o piso
3
>>> 10/-3
-4
>>> 10%3 # Resto de divisão inteira simbolizado por %
1
>>> 10.0/3
3.3333333333333335
```



Operadores aritméticos

- Observações:
 - A precedência dos operadores aritméticos é a usual.
 - Divisão por 0 resultado em um erro de execução grave! Isso também vale para resto da divisão por 0 (%).
 - As operações com inteiros resultados em números não inteiros são TRUNCADOS, ou seja, arredondados para baixo.
 - Pode-se agrupar operações com parênteses recursivamente (não há colchetes ou chaves)
 - $((a+b)*(a+c))*3$



Variáveis

- São nomes dados a áreas de memória
 - Nomes podem ser compostos de algarismos, letras ou _
 - O primeiro caractere não pode ser um algarismo
 - Palavras reservadas (`print`, `if`, `while`, etc) são proibidas
- Exemplos de nomes:

salario	☑ (correto)
aluno01	☑ (correto)
1ano	☒ (incorreto)
_x	☑ (correto)
nota!01	☒ (incorreto)
nota 01	☒ (incorreto)

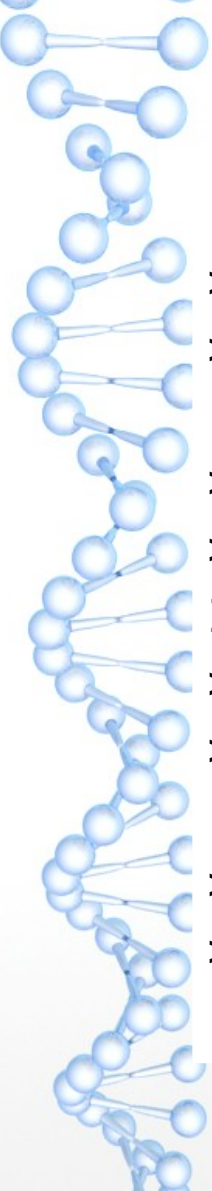
- Servem para:
 - Guardar valores intermediários



Atribuição

- Uma variável é modificada usando o comando de atribuição:
 - **`<var> = <expressão>`**
- É possível também atribuir a várias variáveis simultaneamente:
 - **`var1,var2,...,varN = expr1,expr2,...,exprN`**
- Exemplos:
 - **`nota_maxima = 10.0` *#nota_maxima recebe 10.0***
 - **`c = a` *#c recebe o valor de a***
 - **`b = c+2*a`**
 - **`soma = a+b`**
 - **`fator = 3.0`**
 - **`circunf = 2*pi*raio`**

Em Python



```
>>> a=1
```

```
>>> a
```

```
1
```

```
>>> a=2*a
```

```
>>> a
```

```
2
```

```
>>> a,b=3*a,a
```

```
>>> a,b
```

```
(6, 2)
```

```
>>> a,b=b,a
```

```
>>> a,b
```

```
(2, 6)
```



Tipos de dados

- As variáveis podem conter diferentes tipos
 - **Int**: números inteiros de *precisão fixa* (–? a ?)
 - 1 , 2 , 15 , -19
 - **Long**: números inteiros de *precisão arbitrária*
 - 1L , 10000L , -9999999L
 - **Floats**: números racionais de *precisão variável* (~14 casas)
 - 1.0 , 10.5 , -19000.00005 , 15e-5
 - **Strings**: São cadeias de caracteres
 - “abcd” , “Uma frase.” , “x”
 - **Obs**: Veremos mais sobre *strings* posteriormente no curso

Tipos das Variáveis

- Variáveis são criadas dinamicamente
- O *tipo* de uma variável muda conforme o valor atribuído, i.e., int, float, string, etc.

■ Ex.:

```
>>> a = "1"
```

```
>>> b = 1
```

```
>>> a+b
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in ?
```

```
TypeError: cannot concatenate 'str' and 'int'  
objects
```

ERRO! Identificado
pelo Python



Números inteiros

- `>>> a=2**30 # Potenciação`
- `>>> a`
- `1073741824`
- `>>> b=a*1000 #conversão para long`
- `>>> b`
- `1073741824000L`
- `>>> b/1000`
- `1073741824L`

- `>>> 10.0 # ponto flutuante`
- `10.0`
- `>>> 99e3 #notação científica é opcional`
- `99000.0`
- `>>> 99e-3`
- `0.099`



Expressões booleanas

Também chamadas expressões lógicas

Resultam em verdadeiro (True) ou falso (False)

São usadas em comandos condicionais e de repetição

Servem para analisar o estado e permitir escolher o próximo passo

Operadores mais usados

Relacionais: > , < , ==, !=, >=, <=

Booleanos: and, or, not



Expressões booleanas

```
>>> 1==1
True
>>> 1==2
False
>>> 1==1 or 1==2
True
>>> 1==1 and 1==2
False
>>> 1<2 and 2<3
True
>>> not 1<2
False
>>> not 1<2 or 2<3
True
>>> not (1<2 or 2<3)
False
```



Expressões booleanas

- As constantes **True** e **False** são apenas símbolos convenientes
- Qualquer valor não nulo é visto como verdadeiro enquanto que **0** (ou **False**) é visto como falso
- Operadores relacionais são avaliados antes de **not**, que é avaliado antes de **and**, que é avaliado antes de **or**
- Porém, use parênteses para aumentar a legibilidade do seu código!



Primeiros comandos em Python

Print, input, if else



print

Forma geral: `print expr,expr,...`

Os valores das expressões são escritos um após o outro sem pular de linha:

```
>>> print("1.001 ao quadrado eh ",1.001**2)
1.001 ao quadrado é  1.002001
```

Por exemplo:

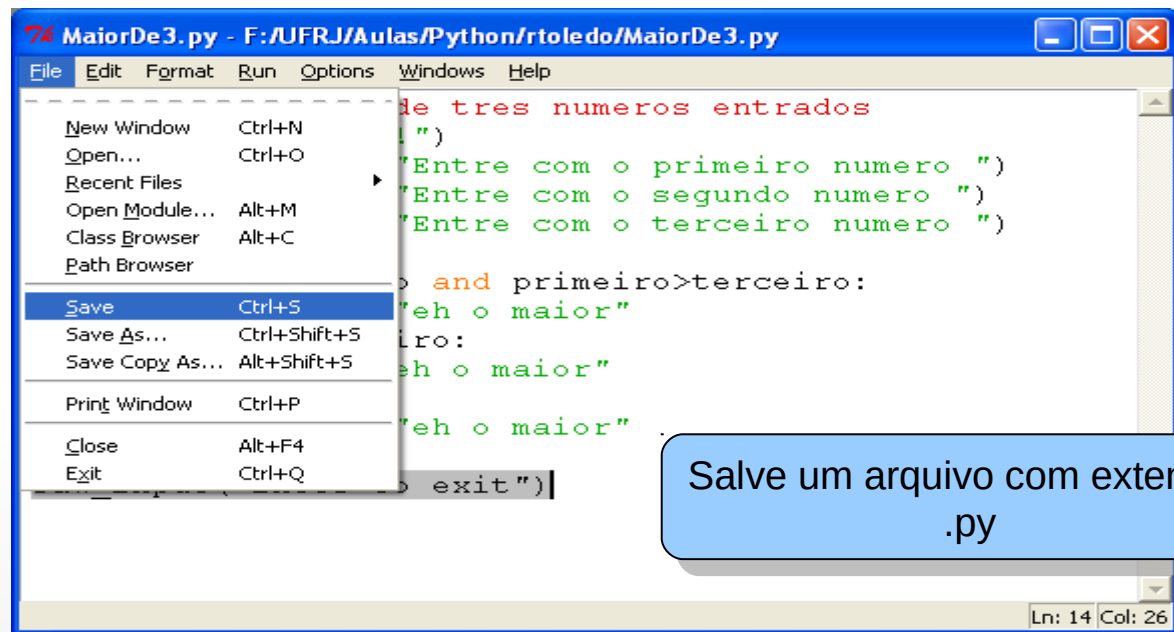
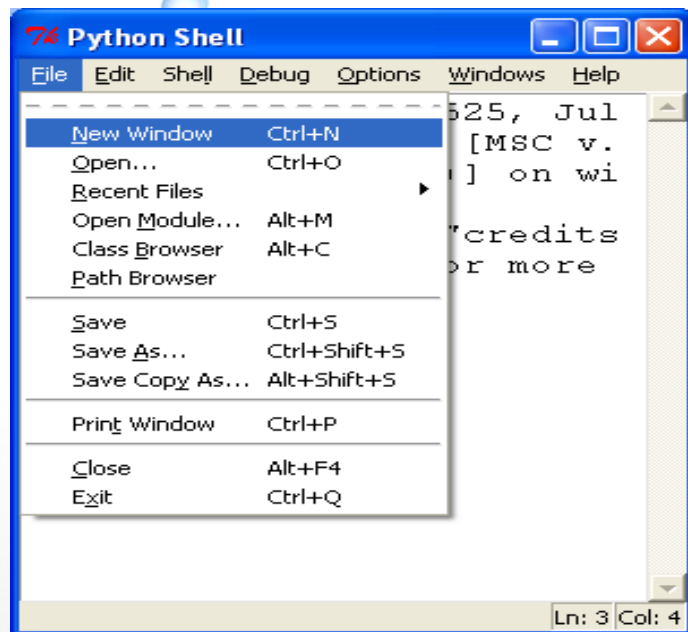
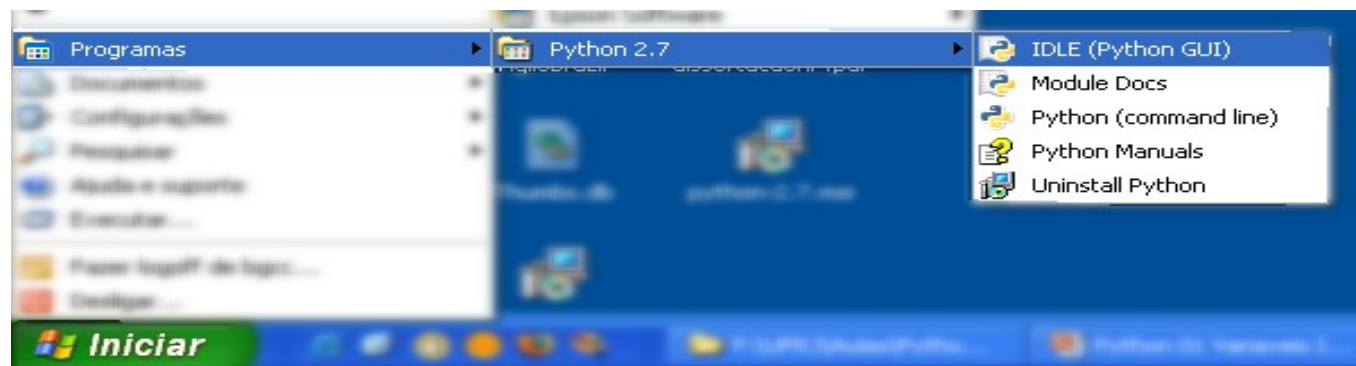
```
>>> aluno = 1234
>>> nota = 8.5
>>> print("A nota do aluno", aluno, "eh", nota)
A nota do aluno 1234 eh 8.50
```



input

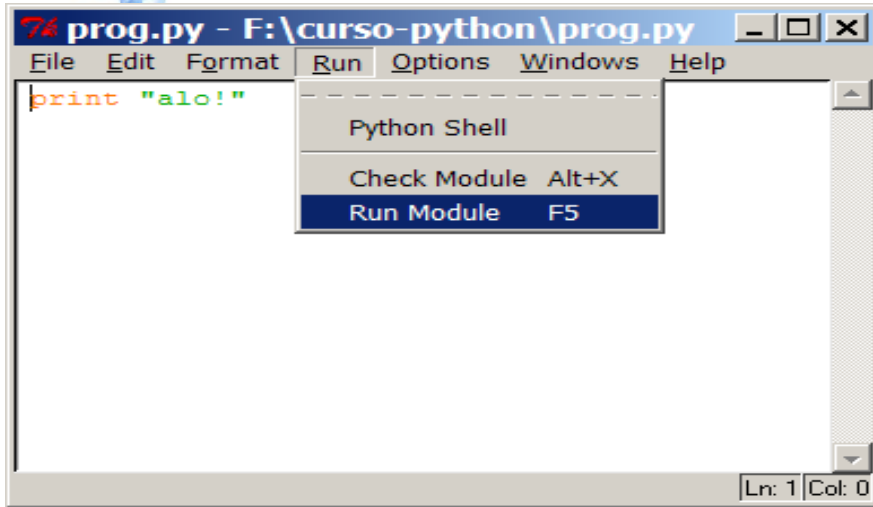
- O comando input permite perguntar ao usuário um valor (normalmente é atribuído a uma variável)
 - Formato: **input**(*pergunta*)
 - onde *pergunta* é uma string opcional que será exibida para indicar o valor que se espera.
- Exemplo:
 - >>> a = input("Entre com um numero: ")
 - Entre com um numero: 19
 - >>> print(a)
 - 19

Criando um .py

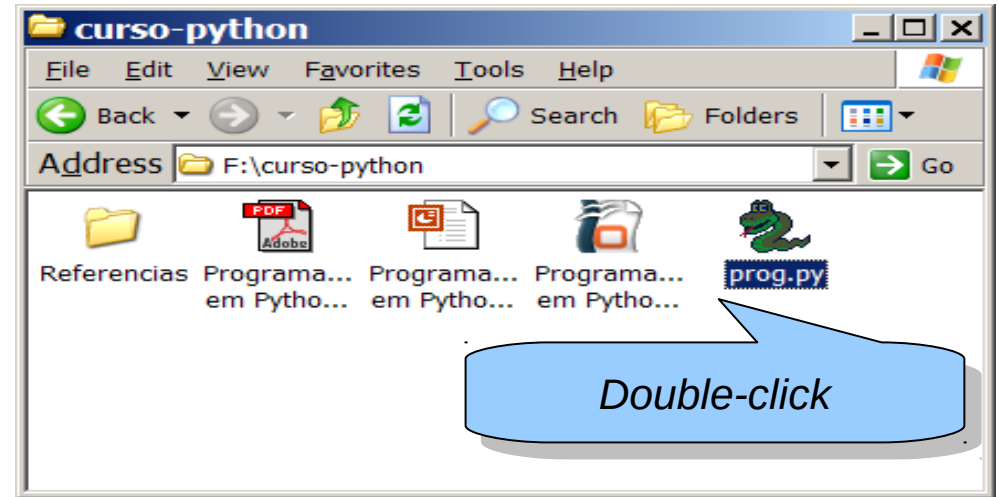


Salve um arquivo com extensão
.py

Executando o .py



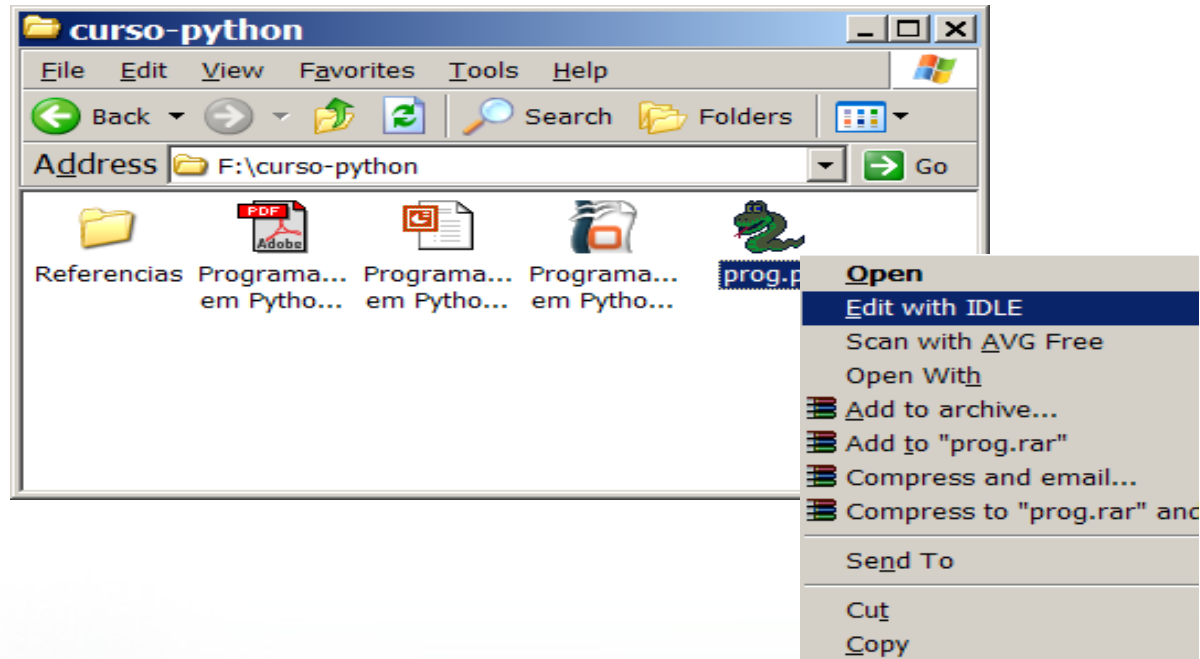
ou...



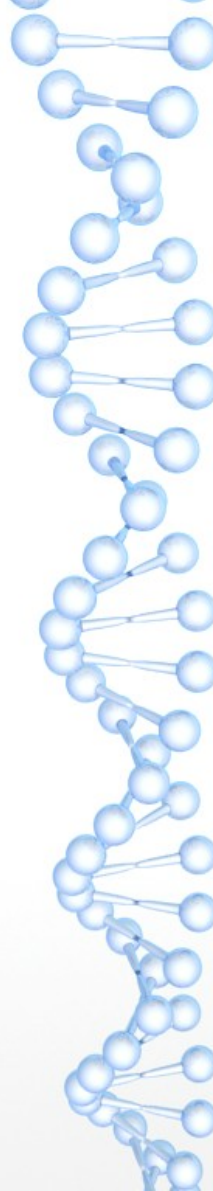
Dica

Para rodar (*double-click*) e ver o resultado final, terminar com
`raw_input("Enter to exit")`

Editando um arquivo já existente de .py



- Ou abrindo pelo IDLE...



Exercícios

(input e print)

- Faça um programa que leia 2 números da entrada e imprima a sua média aritmética.
- Faça um programa que leia um número da entrada e imprima o seu antecedente e o seu sucessor.
- Faça um programa que leia um valor de conta de restaurante, representando o gasto realizado pelo cliente e imprima o valor total a ser pago, considerando que o restaurante cobra 10% para o garçom.
- Faça um programa que leia um valor da hora corrente (hora e minutos) e informe quantos minutos se passaram deste o início do dia.



if

- É o comando condicional por excelência
- Primeira forma:
 - **if** *<expressao>*:
 <bloco de comandos>
- O bloco de comandos será executado apenas se a expressão for verdadeira



if

- Exemplo 1

- `a = input("Entre com um numero:")`
 `if(a < 0):`
 `print (a," é negativo")`
 `print("Obrigado!")`
- Execução 1:
 Entre com um numero:2
 Obrigado!
- Execução 2:
 Entre com um numero:-2
 -2 é negativo
 Obrigado!

Bloco de comandos

- Em python, um bloco de comandos é identificado por uma sequência de comandos indentação, por exemplo 2 espaços.
- Exemplo:
 - `num = input("Entre um numero positivo ")`
 - `if(num>0):`
 - `print ("OK,", num, "eh positivo!")`
 - `num2 = input("Entre com outro numero ")`
 - `if(num2>num):`
 - `print(num2, "eh maior que", num)`
 - `print("A diferença deles eh", num2-num)`
 - `print("acabou!")`

Bloco de comandos

outro



if

- Senão...
- Segunda forma:

– **if** *expressao*:
 comandos1
else:
 comandos2

Executa seq de comandos 1
caso expressão seja
verdadeira.

Caso contrário, executa
seq de comandos 2



if

- Exemplo 2

- `a = input("Entre com um numero:")`
 `if(a < 0):`
 `print(a, " é negativo")`
 `else:`
 `print(a, " é zero ou positivo")`
 `print("Obrigado!")`
- Execução 1:
 Entre com um numero:2
 2 é zero ou positivo
 Obrigado!
- Execução 2:
 Entre com um numero:-2
 -2 é negativo
 Obrigado!



if

- elif...
- Terceira forma:
 - **if** *expressao1*:
 comandos1
 - **elif** *expressao2*:
 comandos2
 - **else**:
 comandos(N)



if

- Exemplo 3

- ```
a = input("Entre com um numero:")
if(a < 0):
 print(a," é negativo")
elif(a==0):
 print(a," é zero")
else:
 print a," é positivo")
print("Obrigado!")
```

- Execução 1:

- Entre com um numero:0  
0 é zero  
Obrigado!

- Execução 2:

- Entre com um numero:2  
2 é positivo  
Obrigado!

# Exercícios (if)

1. Faça um programa que leia um número e diga se ele é par ou ímpar
2. Faça um programa que leia 2 números e imprima uma mensagem dizendo o maior deles. Detalhe: se os números forem iguais, imprima uma mensagem avisando ao usuário.
3. Faça um programa que informe o maior valor de 3 números entrados.
4. Faça um programa que leia três notas de um aluno, calcule sua média aritmética e imprima uma mensagem dizendo se o aluno foi aprovado, reprovado ou deverá fazer prova final. O critério de aprovação é o seguinte:  
aprovado ( $\text{média} \geq 7$ ); reprovado ( $\text{média} < 3$ ) e prova final ( $3 \leq \text{média} < 7$ ).
5. Faça um programa que leia 3 números e imprima uma das seguintes mensagens:  
todos os números são iguais;  
todos os números são diferentes; ou  
apenas dois números são iguais.
6. Faça um programa que leia 3 números e imprima o valor intermediário, entre o menor e o maior número. Suponha que os números serão diferentes.
7. (DESAFIO) Faça um programa que leia 3 números e os imprima em ordem decrescente.
8. (DESAFIO) Um cinema faz descontos no seu preço relativos ao dia da semana e a idade dos clientes. Faça um programa que leia o preço normal de um ingresso, o dia na semana e a idade do cliente e informe o preço final. As datas da semana são representados por números de 1 a 7 e os descontos seguem a tabela abaixo

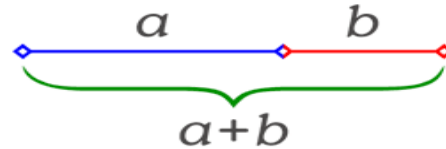


# Python

# While

# Sobre a série de Fibonacci

- Fibonacci, matemático do século XII
- $$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos} \end{cases}$$
- Serie
  - 1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597, ...
- Tendem à razão áurea
  - 1.618033989...
  - Razão da beleza
- Na natureza:
  - Espiral: conchas, bromélias, orelha...
  - Flores: girassol...
  - Crescimento populacional dos coelhos
- Várias propriedades matemáticas
- Muito usado em aprendizado de linguagens de programação







# Fibonacci

```
>>> # Série de Fibonacci
... a,b = 0,1
>>> while(b < 10):
... print(b)
... a,b = b,a+b
...
1
1
2
3
5
8
```



# while

- Repete uma sequência de comandos enquanto uma dada expressão booleana é avaliada como verdadeira

- Formato:

```
while expressão:
 comando
 ...
 comando
```

- Exemplo:

```
>>> i = 10
>>> while(i>8):
... print(i)
... i = i-1
...
10 9
```



# Programas armazenados

- À medida que os programas vão se tornando mais complicados, é mais interessante guardá-los em arquivos e executá-los quando necessário
- Arquivo `fibonacci.py` (use um editor de textos como o do IDLE):

```
Série de Fibonacci:
a, b = 0, 1
while(b < 10):
 print(b)
 a, b = b, a+b
```



# Formas de Executar um Programa

- 4 opções:
  - Digite *python fibo.py* no seu shell (ex: janela DOS), ou
  - Clique no ícone do arquivo, ou
  - De dentro do editor IDLE, selecione *Run Module (F5)*, ou
  - De dentro do interpretador *python*:

```
>>> execfile ("fibo.py")
Entre com um numero 5
1 1 2 3
>>>
```



# Observações

- Existe um outro comando de repetição:
  - FOR
  - (o comando FOR será visto posteriormente)
- Em geral usa-se a variável *i* (abreviação de *integer*) para contagem sequencial de inteiros
- É possível haver *loop* dentro de *loop*
  - Nesse caso recomenda-se usar variáveis diferentes para o controle do loop (ex: *i*, *j*, *k* ...), para que não haja interferência
- Atenção para não criar um loop infinito
  - Para interromper: ctrl+c
  - Exemplo no próximo slide



# Laços

- Como em todo comando de repetição, é importante evitar os chamados “laços” ou “loop infinitos”

- Ex.:

```
>>> i = 10
```

```
>>> while(i>8):
```

```
... print(i)
```

```
... i = i+1
```

```
...
```

```
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

```
25 26 27 28 29 30 31 32 33 ...
```



# Atividades

1. Faça um programa que repita na tela 40 vezes a frase “Aprender Python eh muito facil!”
2. Faça um programa que leia quantas vezes a frase “Aprender Python eh muito facil!” deverá ser repetida, imprima também o número da linha.
3. Faça um programa que leia 40 números somando-os e ao final diga o total
4. Faça um programa que inicialmente pergunte quantos números serão entrados e então calcule o seu somatório e a média.



## Exercícios (*loop*)

1. Faça um programa que leia um numero entre 1 e 10 e imprima a sua tabuada de 1 a 10.
2. Faça um programa que leia um número informando a quantidade de números que serão entrados em sequência posteriormente, então leia os números e, ao final, imprima a média deles
3. Faça um programa que leia uma sequência de 10 números e, ao final, imprima o maior deles.
4. Faça um programa que imprima todas as tabuadas de 1 a 10.





## Exercícios (while)

1. Faça um programa que leia uma série de números inteiros e imprima a sua média, o programa termina com a entrada do número 0.
2. Faça um programa que leia uma série de números positivos e diga ao final qual é o maior (termina com um número negativo).
3. Altere o programa anterior para imprimir também qual é o segundo maior.
4. Faça um programa que leia 2 números da entrada e imprima o resto da divisão inteira do primeiro pelo segundo usando subtrações sucessivas (ou seja, não use comando de resto: %).
5. Altere o programa anterior para que imprima também o resultado da divisão (não use comando de divisão: /).
6. Faça um programa que calcule e imprima o CR do período para os alunos da UFRJ (matrícula inferior ou igual a zero termina a entrada). Para cada aluno, o programa deverá ler:  
número da matrícula; | quantidade de disciplinas cursadas; | Para cada disciplina cursada: | nota da disciplina. | créditos da disciplina  
Obs1: O CR é calculado segundo a média ponderada das notas.  
Obs2: O programa termina com a entrada de uma matrícula negativa.



## Exercício: números primos

- Fazer um programa que decida se um número positivo dado é primo ou não
  - Entrada: número inteiro positivo
  - Saída: diagnóstico de primalidade do número
- Exemplos de execução:
  - Entre com um número inteiro positivo: 169  
169 é múltiplo de 13
  - Entre com um número inteiro positivo: 983  
983 é primo



## Exercício: números primos

- Um número natural é primo se é divisível apenas por si mesmo e pela unidade (excetuando-se o 1)
- Isto sugere o seguinte algoritmo:
  - Se o número é 1, então **não** é primo
  - Se o número é 2, então é primo
  - Caso contrário,
    - Seja  $d$  um possível divisor, cujo valor é inicialmente 2
    - Repetir
      - Se o resto da divisão do número por  $d$  é zero, então o número não é primo
      - Caso contrário, incrementar  $d$
      - Se  $d$  é igual ou maior que o número, então terminar repetição diagnosticando o número como primo



# Resumo até agora

- Em programação, temos os seguintes papéis:
  - cliente, programador, usuário e computador
- Em python:
  - variável: espaço em memória para armazenar conteúdo
  - atribuição = (da direita para esquerda)
  - print: informar ao usuário
  - input: pegar informação do usuário
  - ==, !=, >, >=, <, <=, and, or !
  - if, elif e else
  - while



# Dicas valiosas

- Escolha bons nomes de variáveis
  - Diminui a chance de você se perder
  - Maior legibilidade para o programador futuro
- Foque primeiro em entender qual é o resultado esperado.
  - Ou seja, pense qual será o seu output
- Pense em diferentes testes de entradas
  - Para cada teste de entrada, verifique se a saída está correta



# Matemática

- ***Relação entre dois valores***

- Relação entre **a** e **b**:
  - a) obter os valores de **a** e **b**.
  - b) calcular  $a / b$ .

- ***Percentual entre dois valores***

- Percentual de **a** em relação a **b**:
  - a) obter a relação entre **a** e **b**.
  - b) multiplicar o valor obtido por 100



# Loops (1/3)

- Loop com quantidade determinada em tempo de programação (número de vezes constante)

(O comando FOR é o mais apropriado, porém exemplos com WHILE)

Usar uma variável para contar (ex: `i`)

- Com while, inicializar contador antes do loop e atualizá-lo dentro.
- Exemplo: executar um comando ou um bloco de comandos 100 vezes.

```
i = 0
while i<100
 <comando>
 i = i+1 # i+=1 também funciona, apenas uma
 abreviação...
```



## Loops (2/3)

- Loop com quantidade determinada no início da execução

(O comando FOR é o mais apropriado, porém exemplos com WHILE)

Usar uma variável para contar (ex: *i*)

- Exemplo: executar um comando ou um bloco de comandos *n* vezes, onde *n* é uma variável cujo valor já foi calculado, inicializado ou lido anteriormente:

```
n = input ("Entre com a quantidade de vezes :")
i = 0
while i<n
 <comando>
 i += 1
```





# Loops (3/3)

- Loop com quantidade indeterminada de vezes
  - Inicializar variáveis de controle fora do loop (geralmente imediatamente antes do loop começar)
  - Montar a condição do loop
  - Ao final do bloco de comandos, atualizar as variáveis de controle
    - Exemplo: ler uma seqüência de números inteiros até que o valor digitado seja igual a zero.

```
num = input("Entre com o primeiro numero")
```

```
While num!=0:
```

```
 <comandos>
```

```
 num = input("Entre com o proximo numero")
```



# Loops exemplo

- O chefe do departamento da universidade gostaria que os professores informassem a média da turma a cada prova. Para ajudá-los:  
Faça um programa que leia 40 notas e diga a média.
  - Quem são o cliente, o usuário e o programador?
- As turmas tem tamanho variado, então, modifique o programa anterior para perguntar o total de alunos antes de ler a sequência.
- Os professores reclamaram porque, como alguns alunos faltam a prova, eles são obrigados a contar o total de provas para rodar o programa. Modifique o programa anterior para que o professor não seja obrigado a dizer quantas provas são, o professor pode entrar diretamente com as notas e, ao final, entrará com um valor negativo para indicar o fim da série.

# As 3 soluções

```
i, soma = 0, 0.0
while i<40:
 nota = input("Entre com a proxima nota: ")
 soma = soma+nota
 i = i+1
print("A media eh :", soma / 40)

n = input("Entre com a quantidade de provas: ")
i, soma = 0, 0.0
while i<n:
 nota = input("Entre com a proxima nota: ")
 soma = soma+nota
 i = i+1
print("A media eh :", soma / n)

i, soma = 0, 0.0
nota = input("Entre com a primeira nota: ")
while nota>=0:
 soma = soma+nota
 i = i+1
 nota = input("Entre com a proxima nota: ")
print("A media eh :", soma / i)
```



# Acumulador e Contador

- Inicializar a variável com o valor inicialmente já acumulado (geralmente, imediatamente antes do loop).
- Dentro do loop, atualizar o acumulador
  - Exemplo contador: Leia as notas dos 40 alunos e ao final diga o total em prova final (<7.0)

```
i,pf=0,0
while i<40:
 nota = input("Entre com nota: ")
 if nota<7.0:
 pf +=1
 i+=1
print("Total em prova final: ", pf)
```

- Exemplo acumulador: Leia os 12 faturamentos mensais de uma empresa, indicando o seu faturamento anual.

```
i,anual=0,0
while i<12:
 mensal = input("Entre com faturamento do mes: ")
 anual += mensal #anual = anual + mensal
 i +=1
print("Faturamento anual: ", anual)
```



# Competidor

Cálculo do mais relevante (maior, menor, etc)

- Obter o primeiro elemento da série e armazená-lo como o mais relevante (normalmente antes do loop)
- Realizar um loop obtendo o **restante** da série
  - Para cada elemento obtido, testar se este é mais relevante do que o que já estava previamente armazenado como relevante
- Exemplo: O maior de 10 números entrados

```
maior = input("Entre com o primeiro numero")
i=1
while i<=10:
 num = input("Entre com o proximo numero")
 if num > maior
 maior = num
 i = i+1
print(maior)
```



## Competidor (continuação)

- Exemplo: O maior de 10 números entrados, **mas em qual posição?**

```
maior = input("Entre com o primeiro numero")
i=1
posicao = 1
while i<=10:
 num = input("Entre com o proximo numero")
 if num > maior
 maior = num
 posicao = i
 i = i+1
print("O maior eh", maior, "na posição", posicao)
```



# Python

- Além do comando `while` recém apresentado, Python tem as estruturas usuais de controle de fluxo conhecidas em outras linguagens, com algumas particularidades.
- O comando `for` em Python difere um tanto do que você talvez esteja acostumado em C ou Pascal. Ao invés de se iterar sobre progressões aritméticas (como em Pascal), ou dar ao usuário o poder de definir tanto o passo da iteração quanto a condição de parada (como em C), o comando `for` de Python itera sobre os itens de qualquer sequência (como uma lista ou uma string), na ordem em que eles aparecem na sequência. Por exemplo:



# Python

```
>>> # Medir o tamanho de algumas strings:
```

```
>>> a = ['gato', 'janela', 'defenestrar']
```

```
>>> for x in a: ... print x, len(x) ... gato 4 janela 6 defenestrar 11
```

```
>>>
```

Não é seguro modificar a sequência sobre a qual se baseia o laço de iteração (isto pode acontecer se a sequência for mutável, isto é, uma lista). Se você precisar modificar a lista sobre a qual está iterando (por exemplo, para duplicar itens selecionados), você deve iterar sobre uma cópia da lista ao invés da própria. A notação de fatiamento é bastante conveniente para isso:

```
>>> for x in a[:]: # fazer uma cópia da lista inteira
```

```
... if len(x) > 6: a.insert(0, x)
```

```
...
```

```
>>> a
```

```
['defenestrar', 'gato', 'janela', 'defenestrar']
```





# Python

A função `range()`

Se você precisar iterar sobre sequências numéricas, a função embutida `range()` é a resposta. Ela gera listas contendo progressões aritméticas, por exemplo:

```
>>> range(10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

O ponto de parada fornecido nunca é incluído na lista; `range(10)` gera uma lista com 10 valores, exatamente os índices válidos para uma sequência de comprimento 10. É possível iniciar o intervalo em outro número, ou alterar a razão da progressão (inclusive com passo negativo):



# Python

```
>>> range(5, 10)
```

```
[5, 6, 7, 8, 9]
```

```
>>> range(0, 10, 3)
```

```
[0, 3, 6, 9]
```

```
>>> range(-10, -100, -30)
```

```
[-10, -40, -70]
```



# Python

Para iterar sobre os índices de uma sequência, combine `range()` e `len()` da seguinte forma:

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
```

```
>>> for i in range(len(a)):
```

```
... print(i, a[i])
```

```
...
```

```
0 Mary
```

```
1 had
```

```
2 a
```

```
3 little
```

```
4 lamb
```



# Python

Comandos `break` e `continue`, e cláusulas `else` em laços

- O comando `break`, como em C, interrompe o laço `for` ou `while` mais interno.
- O comando `continue`, também emprestado de C, avança para a próxima iteração do laço mais interno.
- Laços podem ter uma cláusula `else`, que é executada sempre que o laço se encerra por exaustão da lista (no caso do `for`) ou quando a condição se torna falsa (no caso do `while`), mas nunca quando o laço é interrompido por `break`. Isto é exemplificado no próximo exemplo que procura números primos:



# Python

```
>>> for n in range(2, 10):
... for x in range(2, n):
... if n % x == 0:
... print(n, '=', x, '*', n/x)
... break
```

(Sim, este é o código correto. Olhe atentamente:  
a cláusula else pertence ao laço for, e **não** ao comando if.)

```
... else:
... # laço terminou sem encontrar um fator
... print(n, 'é um número primo')
...
2 é um número primo
3 é um número primo
4 = 2 * 2
5 é um número primo
```



# Strings

- São cadeias de caracteres
- Constituem outro tipo fundamental do python
- Constantes ***string*** são escritas usando aspas simples ou duplas
  - Ex.: "string" ou 'string'
- O operador "+" pode ser usado para concatenar strings
  - Ex.: "a"+"b" é o mesmo que "ab"
- O operador \* pode ser usado para repetir strings
  - Ex.: "a"\*10 é o mesmo que "aaaaaaaaaa"



# Output e Input de strings

## Verifique a versão do python

- print

- Ex:

```
>>> frase = "Exemplo de frase"
>>> print(frase)
Exemplo de frase
```

- input

- Com input, só é possível ler uma frase (ou mesmo um caracter) se essa for digitada entre aspas.

- Ex 1:

```
>>> frase = input("Entre a frase: ")
Entre a frase: teste
Traceback (most recent call last):
 File "<pyshell#52>", line 1, in <module>
 frase = input("Entre a frase: ")
 File "<string>", line 1, in <module>
NameError: name 'teste' is not defined
```

- Ex 2:

```
>>> frase = input("Entre a frase: ")
Entre a frase: "Teste"
>>> print(frase)
Teste
>>> frase
'Teste'
```



## raw\_input

### Verificar a versão do python

- É semelhante ao `input`, mas não tenta interpretar o que foi digitado como uma expressão
  - O resultado é simplesmente uma string com o texto digitado
  - Ex.:

```
>>> nome = raw_input("Entre com o seu nome: ")
Entre com o seu nome: Rodrigo de Toledo
>>> nome
'Rodrigo de Toledo'
```
- Ou seja, **use o RAW\_INPUT** para ler strings





## (uma observação sobre o print)

- Se o comando terminar com vírgula, o próximo print escreverá na mesma linha.
- Por exemplo:

```
>>> a, b = 0, 1
>>> while b < 1000:
... print(b)
... a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```



# Exercícios

- Faça um programa que leia todas as notas da p1 de uma turma de 50 alunos. Para cada aluno deverá ser lido o seu nome, o sexo ('M' ou 'F') e a nota. Ao final diga quantos por cento da sala são mulheres; qual foi a média dos homens e a média das mulheres; qual homem e qual mulher tiraram as maiores notas do seu respectivo gênero.
- Faça um programa que controle o painel da pontuação de uma partida de basquete. **Inicialmente, leia os nomes dos times de basquete** e a cada cesta o usuário deverá informar qual o time realizou o arremesso (**A ou B**) e de quantos pontos foi a cesta. O usuário pode entrar com 0 pontos, caso tenha se enganado de time ou o lance tenha sido anulado. Caso o usuário entre com uma pontuação inválida (negativa ou maior que 3) o programa deverá pedir para que entre novamente com a pontuação. A cada cesta o placar deve ser informado. O programa termina com time inválido quando deverá informar o time vencedor e o placar final.



# Strings – Índices

- Endereçam caracteres individuais de uma string
  - Notação: *string*[*índice*]
  - O primeiro caractere tem índice 0
  - O último caractere tem índice -1
  - Ex.:

```
>>> frase = "Esta eh uma frase!"
>>> frase[0]
'E'
>>> frase[-1]
'!'
>>> frase[4]
' '
```



# Percorrendo os caracteres de uma string

## Verifique a versão do python (input | raw\_input)

- Para saber o tamanho de uma string, usa-se o comando `len(<string>)`
- Para percorrer, podemos usar o `while` (fica mais fácil usar o comando `for`, como veremos mais tarde)

- Exemplo:

O que faz o programa abaixo?

```
letra = raw_input("Entre com uma letra: ")
frase = raw_input("Entre com uma frase: ")
cont = 0
i=0
while i<len(frase):
 if frase[i] == letra:
 cont+=1
 i+=1
print(cont)
```

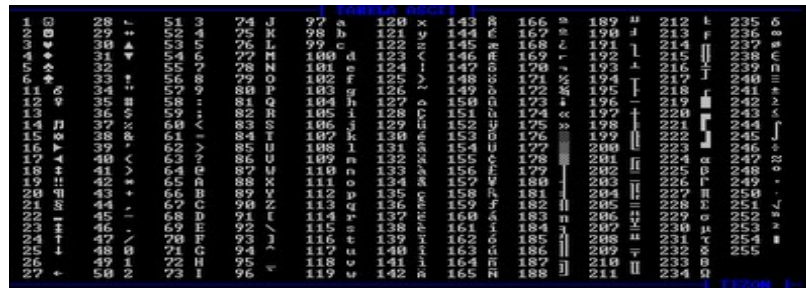
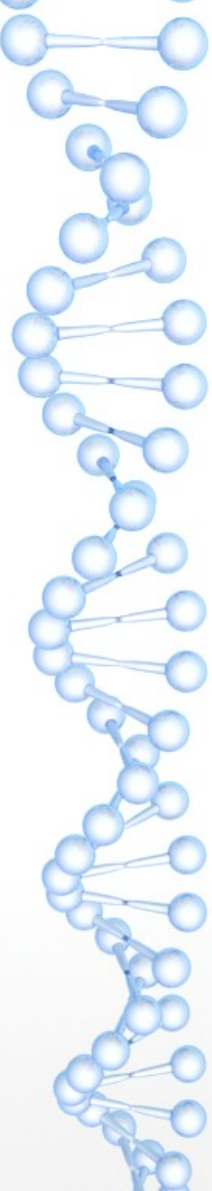


# Exercícios

- Faça um programa que conte a ocorrência de um determinado caracter (lido inicialmente) em um texto (conjunto de linhas terminadas por uma linha vazia), imprimindo o total.
- Modifique o programa acima para que procure pelo caracter no texto, indicando a cada linha a(s) posição(ões) onde o caracter aparece. Ao final, indique a linha em que houve maior ocorrência de caracter.
- Faça um programa que leia um texto (conjunto de linhas terminadas por <Enter>), lido da entrada, indicando qual é a menor linha, qual a quantidade total de algarismos digitados e em qual linha se encontra a maior seqüência consecutiva de algarismos.

# ASCII

- Python usa a tabela de caracteres default do S.O.
  - Ex.: ASCII, UTF-8
- Tabela descrevendo os 256 caracteres representáveis, de 0 a 255 .
  - (1caracter = 1byte = 8 bits = 256).
- Nela estão contidos:
  - Letras (maiúsculas ou minúsculas),
  - Algarismos,
  - Símbolos e
  - Caracteres especiais (enter, backspace, tabulação...)
- Informações úteis sobre a tabela ASCII:
  - As letras minúsculas estão consecutivas em ordem alfabética.
  - As letras maiúsculas estão consecutivas em ordem alfabética.
  - Os algarismos estão consecutivos em ordem crescente.



|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 | 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 | 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|



# caracteres especiais

- Caracteres não imprimíveis podem ser expressos usando notação “barra-invertida” (\)
  - \n é o mesmo que *new line*
  - \t é o mesmo que *tab*
  - \r é o mesmo que *carriage return*
  - \b é o mesmo que *backspace*
  - \\ é o mesmo que \
  - \x41 caracter cujo código hexadecimal é 41 (“A” maiúsculo)

```
>>> print("hello\nworld")
hello
world
>>> print("hello\tworld")
hello world
>>> print("hello\bworld")
helloworld
>>> print("hello\rworld")
helloworld
>>> print("hello\\world")
hello\world
>>> print("hello\x41world")
helloAworld
```





# Strings

```
>>> print "ab\rd" # print exibe chars não imprimíveis
db
>>> print "abc\td"
abc d
>>> print "abc\nd"
abc
d
>>> print "abc\\nd"
abc\nd
>>> print "ab\bc"
ac
>>> print "\x41\xA1"
Aí
```





# Strings

- Também é possível escrever constantes string em várias linhas incluindo as quebras de linha usando três aspas como delimitadores

- Ex.:

```
>>> print("""
Um tigre
dois tigres
três tigres""")
```

```
Um tigre
dois tigres
três tigres
```

```
>>> print('''abcd
efgh''')
```

```
abcd
efgh
```



# Exemplo

Leia uma string e conte o total de letras minúsculas, letras maiúsculas e algarismos.

```
frase = input("Entre com uma frase: ")
minu, maiu, alga = 0,0,0
i=0
while i<len(frase):
 if "a" <= frase[i] <= "z":
 minu+=1
 elif "A" <= frase[i] <= "Z":
 maiu+=1
 elif "0" <= frase[i] <= "9":
 alga+=1
 i+=1
print("Minusculas:", minu)
print("Maiusculas:", maiu)
print("Algarismos:", alga)
```



# Exercícios

1. Leia uma string e retorne o número de palavras encontradas:
  - a) Considere que serão entradas apenas letras e sempre um único espaço entre palavras
  - b) Considere os caracteres '.', ',', ';' como separadores de palavras, além do espaço em branco.
  - c) Considere que cada palavra pode estar separada por um ou mais caracteres de espaço.
2. Faça um programa que leia uma string e verifique se é um palíndromo, isto é, da esquerda para direita ou da direita para a esquerda têm-se a mesma sequência de caracteres. Exemplos: ama, ovo, asa, erre, arara.
3. DESAFIO:
  - Altere o programa anterior para que funcione considerando apenas as letras. Por exemplo, a frase “socorram-me, subi no onibus em marrocos” deve ser considerado uma palíndrome.



# Strings – Fatias (slices)

- Notação para separar trechos de uma string
  - Notação: `string[índice1:índice2]`
  - Retorna os caracteres desde o de índice1 (inclusive) até o de índice2 (exclusive)
  - Se o primeiro índice é omitido, é assumido 0
  - Se o último índice é omitido, é assumido o fim da string



# Strings – Fatias (slices)

```
>>> a = 'abcde'
>>> a[0:2]
'ab'
>>> a [2:]
'cde'
>>> a[:]
'abcde'
>>> a[-1:]
'e'
>>> a[:-1]
'abcd'
>>> a [3:1]
''
>>> a [-1:-3]
''
```



# Exercícios

1. Faça um programa que leia um caracter e uma frase. Em seguida imprima apenas o trecho da frase entre a primeira e a última ocorrência do caracter entrado, excluindo início e fim. Caso só haja uma ocorrência, imprima apenas o caracter. Caso haja nenhuma ocorrência, apenas informe isso ao usuário!
2. Faça um programa que leia uma string e a imprima segundo a seguinte regra:
  - Se a string tiver mais de 40 caracteres, ela deve ser particionada, quantas vezes for necessário, de modo a imprimir no máximo 40 caracteres por linha, sendo que nenhuma palavra deve ser impressa fragmentada.



# Exercício: algarismos romanos

- Fazer um programa que escreva a representação em algarismos romanos de um número inteiro positivo
  - O usuário deve entrar com um número (*input*)
  - O resultado deve ser impresso no console (*print*)
- Exemplo de execução:

Entre com um numero positivo: 1985

Em algarismos romanos: MCMLXXXV



# Exercício: algarismos romanos

- Algoritmo
  - » A representação em romanos é uma string à qual é acrescentada uma letra por vez
- Inicialmente, uma string vazia
  - » Examinar as sucessivas potências de 10
- Por exemplo, a letra 'M' corresponde à casa dos milhares
- Se o número é 2200, sabemos que teremos dois M's na representação em romanos
- Sabemos que há M's se o número é maior ou igual a 1000
- Sempre que um milhar for computado, subtrair 1000 do número
  - » Um processamento semelhante é feito para outros algarismos romanos, por exemplo:
- Se o número é maior ou igual que 500, acrescentar 'D'
- Se o número é maior que 900, acrescentar 'CM'





# Exercício: algoritmos romanos

- DICA: processando um número entre 1 e 9

```
if num >= 9:
 romano = romano + "IX"
 num = num-9
if num >= 5:
 romano = romano + "V"
 num = num-5
if num >= 4:
 romano = romano + "IV"
 num = num - 4
while num >= 1:
 romano = romano + "I"
 num = num - 1
```



# Evitando repetições

- Em computação sempre queremos evitar repetições.
- Toda vez que fazemos copy&paste de algum código, é provável que estejamos programando mal.
- Várias linguagens de programação incluem o conceito de subprograma (ou subrotina)
  - Atribui-se um nome à uma seqüência de comandos, e faz-se referência a este nome nos vários lugares do programa onde a seqüência em questão deveria ser repetida.
- Em Python, sub-programas têm o nome de **funções**
- Formato geral:

```
def nome (arg, arg, ... arg):
 comando
 ...
 comando
```



# Motivação

- Faça um programa que apresente um menu com as seguintes opções: (a)xxxx, (b)yyyy, (c)www, (d)zzzz e (e)Sair. Quando for a.....
- ```
print ((a)xxxx\n(b)yyyy\n(c)www\n(d)zzzz\n(e)Sair")
opcao = input("Entre com a opcao: ")
while opcao != "e":
    ...
    print("(a)xxxx\n(b)yyyy\n(c)www\n(d)zzzz\n(e)Sair")
    opcao = input("Entre com a opcao: ")
```
- Garanta que o usuário entrou com a opção certa, senão leia novamente.
- Para não repetir tudo isso duas vezes, vamos colocar numa função



Retornando um valor

- Uma função pode retornar um valor. Exemplo:
 - ```
def menu():
 print("(a)xxxx\n(b)yyyy\n(c)www\n(d)zzzz\n(e)Sair")
 letra = input("Entre com a opcao: ")
 while letra < "a" or letra > "e":
 print("Opcao invalida!")
 letra = input("Entre com a opcao novamente: ")
 return letra

opcao = menu()
while(opcao != "e"):
 ...
 opcao = menu()
```
- Observar:
  - Comando return
  - A função tem que ser definida antes de ser usada
    - Por exemplo, digitada no alto do arquivo .py



# Vantagens de evitar repetições

- Menos trabalho
- Mais legível
- Dividir grandes tarefas de computação em tarefas menores
  - permite pensar num problema em diversos níveis
  - modularização
- Menos chances de bugs
- Facilita a manutenção
- Código menor
- .



# Definindo funções

- Em Python, sub-programas têm o nome de ***funções***
- Formato geral:

```
def nome (arg, arg, ... arg):
 comando
 ...
 comando
```
- Onde:
  - *nome* é o nome da função
  - *args* são especificações de argumentos da função
    - Uma função pode ter nenhum, um ou mais argumentos
  - *comandos* contêm as instruções a serem executadas quando a função é invocada



# Resultado de funções

- Uma função tipicamente computa um ou mais valores
- Para indicar o valor a ser devolvido como o resultado da função, usa-se o comando `return`, que tem o formato `return expressão`
  - onde a *expressão* é opcional e designa o valor a ser retornado
- Ao encontrar o comando `return`, a função termina imediatamente e o controle do programa volta ao ponto onde a função foi chamada
  - observe que pode haver mais de um `return` dentro da função
- Se uma função chega a seu fim sem nenhum valor de retorno ter sido especificado, o valor de retorno é `None`



# Exemplo

```
>>> def f():
 return
```

```
>>> print f()
None
```

```
>>> def f():
 return "Oi"
```

```
>>> print f()
Oi
```

```
>>> def f(nome):
 return "Oi, " + nome + "!"
```

```
>>> print f("Joao")
Oi, Joao!
```





# Argumentos de funções

- Argumentos (ou parâmetros) são como variáveis que recebem seus valores iniciais do chamador
- Essas variáveis, assim como outras definidas dentro da função são ditas *locais*, isto é, só existem no lugar onde foram definidas
  - Ao retornar ao ponto de chamada, as variáveis locais são descartadas
- Se uma função define  $n$  argumentos, a sua chamada deve incluir valores para todos eles
  - Exceção: argumentos com valores default



# Exemplo

```
>>> def quadrado(x):
 return x*x
```

```
>>> print(quadrado(10))
100
```

```
>>> print(x)
```

```
.....
```

```
NameError: name 'x' is not defined
```

```
>>> print(quadrado())
```

```
.....
```

```
TypeError: quadrado() takes exactly 1 argument (0
given)
```



# Função chamando função

- Uma função pode ser chamada por outra função, e assim por diante. Ou seja, existe uma pilha de chamadas de função (*call stack*).
  - Altere a função menu para que permita que o usuário entre com minúsculas ou maiúsculas
- ```
- def minusc(letrinha):  
    if("A" <= letrinha <= "Z"):  
        return letrinha-"A"+"a"  
    return letrinha  
  
- def menu():  
    print("(a)xxxx\n(b)yyyy\n(c)www\n(d)zzzz\n(e)Sair")  
    letra = minusc(raw_input("Entre com a opcao: "))  
    while(letra < "a" or letra > "e"):  
        print("Opcao invalida!")  
        letra = minusc(raw_input("Entre com a opcao novamente: "))  
    return letra
```



Variáveis Locais

- As variáveis de uma função NÃO tem NENHUMA relação com as variáveis da outra (o programa também é uma função). Ou seja, se você mencionar uma variável na função1 declarada na função2 irá dar erro. Outra consequência é que podem existir variáveis com mesmo nome em funções diferentes e nada terão em comum (além do nome).
- O valor de uma variável que pertença a uma determinada função morre quando essa função termina. Ou seja, toda vez que uma função é chamada é como se ela estivesse sendo executada pela primeira vez.
- Argumentos também são variáveis locais e valem as mesmas regras.
- Evitem variáveis globais!



Alterando parâmetros

- É possível alterar parâmetros?
 - Sim e não
 - Como o parâmetro é uma variável local, ele pode ser alterado sem problemas
 - Entretanto, se um parâmetro recebe um valor que vem de uma variável, esta não é alterada
- Ex.:

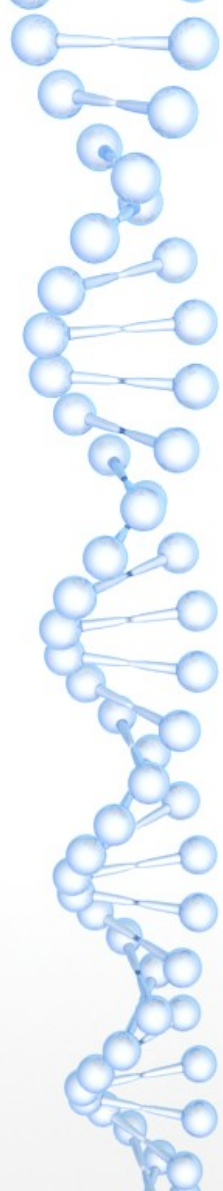
```
>>> def quadrado(x):  
    x = x*x  
    return x  
  
>>> x = 10  
>>> print quadrado(x)  
100  
>>> print x  
10
```



Alterando parâmetros (alterando conteúdo de listas)

- Note que quando passamos uma variável do tipo lista como parâmetro, estamos passando uma *referência* para um valor do tipo lista
 - Nesse caso, alterar o parâmetro pode influenciar no “valor” do conteúdo da lista
 - Exemplo:

```
>>> def f(x):  
        x[:] = [5]  
  
>>> a = [1,2,3]  
>>> f(a)  
>>> a  
[5]
```



Retorno de mais de um valor

- Python permite o retorno de mais de um valor.

- Exemplo:

```
>>> def minmax(x,y):  
    if x<y:  
        return x,y  
    else:  
        return y,x
```

```
>>> a = 5  
>>> b = 10  
>>> menor,maior = minmax(a,b)  
>>> menor  
5  
>>> maior  
10
```



Observações

- A declaração da função deve ser feita antes da sua chamada na outra função para que o Python reconheça o seu nome.
- Os valores passados como parâmetros na função chamadora, serão recebidos pela função chamada **exatamente** na mesma ordem.
- Uma função pode chamar a si própria, isso é chamado de função recursiva. Cada vez que a função é chamada, uma nova instância de suas variáveis e parâmetros é criada.
- Não pode haver funções e variáveis com o mesmo nome!



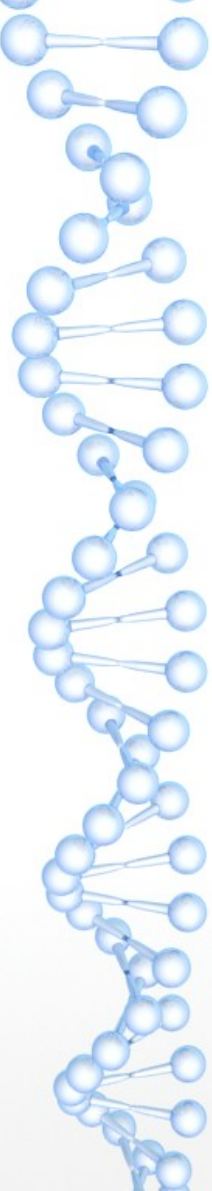
Exercício

- Altere programas que usavam menu para que sejam usados com função!
 - Exercícios já passados com menu:
 - Hortifruti(esse é o mais interessante para treinar)
 - Farmácia
 - Alunos de Python
 - UPA (esse tem na página)
 - Hotel (P2 2010.2)
 - Faça uma função para receber a opção do usuário
 - **Atenção:** para cada opção, faça uma função diferente, passando os parâmetros necessários
 - Exemplo:
 - ```
opcao = menu()
while(opcao != "e"):
 if(opcao == "a"):
 incluir(alunos)
 elif (opcao == "b"):
 excluir(alunos)
 ...
```



# Exercício

- Faça o seguinte:
  - Faça uma função que receba um valor inteiro e retorne a divisão por 10 e o resto dessa divisão.
  - Faça uma função que receba um número inteiro entre 0 e 9 e retorne uma string com o nome do algarismo por extenso.
  - Faça um programa que imprima invertido os nomes do algarismos de um número inteiro. (Use suas funções!)
    - Ex: 234      saída: quatro três dois
- Faça uma função que calcule o peso ideal de uma pessoa. A função deve receber o valor da altura da pessoa e um valor numérico indicando o sexo (0 - masculino e 1 - feminino). A função deve calcular o peso ideal de acordo com as seguintes fórmulas: homens →  $(72.7 * altura) - 58$  e mulheres →  $(62.1 * altura) - 44.7$ .
  - Faça um programa para ler a altura e o sexo de um conjunto de pessoas e imprimir o peso ideal até que seja digitado um valor negativo para altura.



# **import (módulos)**



# Módulos?

- Você viu como reutilizar código através do uso de funções. Como fazer para usar suas funções em outros programas que você escrever?
  - Resposta: **Módulos**
- Um módulo basicamente é um arquivo contendo funções e variáveis que você definiu. Para reutilizar o módulo em outros programas o arquivo deve ter uma extensão .py.
- Um módulo pode ser importado por outro programa para que este possa fazer uso da sua funcionalidade.
- Python fornece uma biblioteca padrão com inúmeras funções em diversos módulos já preparadas para uso.

# Importando um módulo

■ exemplo....import X

- Importa o módulo X, e cria uma referência para esse módulo no namespace atual.
- Você pode usar X.nome para se referenciar a funções definidas no módulo X.

■ from X import \*

- Semelhante a instrução anterior, porém todas as definições dentro de X são criadas diretamente no namespace atual.
- Você pode usar simplesmente nome para se referenciar a funções definidas no módulo X.

■ from X import f1, f2, f3

- Comportamento igual ao anterior, mas apenas as funções f1, f2 e f3 são acessíveis
- Isso é especialmente útil para módulos com muitas funções por dois motivos:
  - Evita conflito de nomes com o namespace atual
  - Reduz tempo de carga do módulo

# Criando seu próprio módulo

- Criar seus próprios módulos é fácil, você já está fazendo isso o tempo todo! Cada programa Python é também um módulo. Você apenas tem que se certificar que tem uma extensão .py

Obs: o módulo tem que estar na mesma pasta do programa ou em um diretório conhecido do Python...

```
Filename: mymodule.py
def sayhi():
 print("Hi, this is mymodule.")

version = "0.1"
End of mymodule.py
```

```
Filename: mymodule_demo.py
import mymodule

mymodule.sayhi()
print("Version", mymodule.version)
```

```
Filename: mymodule_demo2.py
from mymodule import sayhi, version
Alternative:
from mymodule import *

sayhi()
print("Version", version)
```



# Exemplo de um módulo

## Verifique a versão do python... print (“xxx”)

■ menus.py

```
#Imprime em forma de menu uma lista de strings
def print_lista (opcoes):
 num = 1
 for op in opcoes:
 print num, "-", op
 num+=1
 opcao = input("Entre com a opcao: ")
 return opcao

#CHAME ESTA FUNCAO QUANDO QUISEER USAR UM MENU NUMERICO
#Retorna apenas quando o usuario entrar uma opcao valida!
def menu_numerico(opcoes):
 opcao = print_lista(opcoes)
 while opcao<=0 or opcao>len(opcoes):
 print "Opcao invalida!"
 opcao = print_lista(opcoes)
 return opcao
```



# Exercício Calculadora

- Faça um programa que leia dois números (exemplo: a, b) na entrada e então apresente o seguinte menu:
  - 1-Soma
  - 2-Subtração (a-b)
  - 3-Multiplicação
  - 4-Divisão (a/b, decimais)
  - 5-Potenciação ( $a^b$ )
  - 6-Fatorial de ambos
  - 7-Inserir novos números
  - 8-Sair
- Obs1: Use a função `menu_numerico` do módulo `menus.py`
- Obs 2: Use funções recursivas (que chamam a si mesmas) para a potenciação e para o fatorial





# Chaves vs. Índices

- Considere que queiramos representar um caderno de telefones
  - Uma solução é ter uma lista de nomes e outra de telefones
    - Telefone de nome[i] armazenado em telefone[i]
    - Acrescentar “Joao” com telefone “20122232”:  
nome+= “Joao” telefone+=“20122232”
    - Para encontrar o telefone de “Joao”:  
tel = telefone[nome.index(“Joao”)]
  - Dicionários tornam isso mais fácil e *eficiente*  
telefone[“Joao”] = “20122232”  
tel = telefone[“Joao”]



# Dicionários

- São estruturas de dados que implementam *mapeamentos*
- Um mapeamento é uma coleção de associações entre pares de valores  
O primeiro elemento do par é chamado de *chave* e o outro de *conteúdo*
  - *chave1* ↔ *conteudo1*
  - *chave2* ↔ *conteudo2*
  - *chave3* ↔ *conteudo3*
- De certa forma, um mapeamento é uma generalização da ideia de acessar dados por índices, exceto que num mapeamento os índices (ou chaves) podem ser de qualquer tipo *imutável*



# Criando dicionários

- Uma constante do tipo dicionário é escrita  
`{ chave1:conteúdo1, ... chaveN:conteúdoN }`
- Uma variável do tipo dicionário pode ser “indexada” da maneira habitual, isto é, usando colchetes com a chave que se deseja acessar.
  - `Teljoao = telefone[“Joao”]`
- O conteúdo associado a uma chave pode ser alterado atribuindo-se àquela posição do dicionário
  - `telefone[“Joao”] = “5555-0000” #novo telefone`
- Novos valores podem ser acrescentados a um dicionário fazendo atribuição a uma chave ainda não definida
  - `telefone[“Maria”] = “5555-0001” #nova amiga`



# Exemplo

```
>>> dic = {}
>>> dic["joao"] = 100
>>> dic
{'joao': 100}
>>> dic["maria"] = 150
>>> dic
{'joao': 100, 'maria': 150}
>>> dic["joao"]
100
>>> dic["maria"]
150
>>> dic["pedro"] = 10
>>> dic
{'pedro': 10, 'joao': 100, 'maria': 150}
```



# Exemplo

– Observe que não há ordem definida entre os pares chave/conteúdo de um dicionário

```
>>> meses = {"janeiro":31, "fevereiro":28, "marco":31, "abril":30,
 "maio":31, "junho":30, "julho":31, "agosto":31, "setembro":30,
 "outubro":31, "novembro":31, "dezembro":31 }
>>> for i in meses:
 print i, meses[i]
novembro 31
marco 31
julho 31
agosto 31
fevereiro 28
junho 30
dezembro 31
janeiro 31
abril 30
maio 31
outubro 31
setembro 30
```



# Dicionários não têm ordem

- As chaves dos dicionários não são armazenadas em qualquer ordem específica
  - Na verdade, dicionários são implementados por tabelas de espalhamento (*Hash Tables*)
  - A falta de ordem é proposital
- Diferentemente de listas, atribuir a um elemento de um dicionário não requer que a posição exista previamente

`X = []`

`X [10] = 5    # ERRO!`

`. . .`

`Y = {}`

`Y [10] = 5    # OK!`



# Percorrendo Dicionários

```
>>> meses = {"janeiro":31, "fevereiro":28, "marco":31,
 "abril":30, "maio":31, "junho":30, "julho":31, "agosto":31,
 "setembro":30, "outubro":31, "novembro":31, "dezembro":31 }
>>> for i in meses:
 print i, meses[i]
```

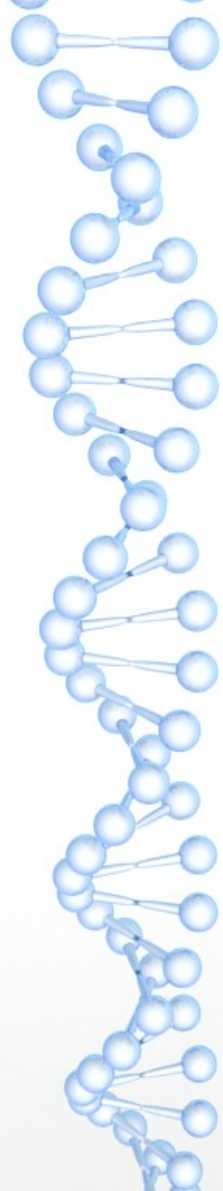


# Métodos *keys* e *values*

- `keys()` retorna uma lista com todas as chaves do dicionário
- `values()` retorna uma lista com todos os valores do dicionário
- Ex.:

```
dic = {'pedro': 10, 'joao': 100, 'maria': 150}
>>> dic.keys()
['joao', 'maria', 'pedro']
>>> dic.values()
[100, 150, 10]
```





# Método *has\_key*

- `has_key(chave)`
  - `dic.has_key(chave)` é o mesmo que `chave in dic`
  - Ex.:

```
>>> dic = { "Joao": "a", "Maria": "b" }
>>> dic.has_key("Joao")
True
>>> dic.has_key("Pedro")
False
```



# Método *pop*

- **pop (chave)**
  - Obtém o valor correspondente a chave e remove o par chave/valor do dicionário
  - Ex.:

```
>>> d = {'x': 1, 'y': 2}
>>> d.pop('x')
1
>>> d
{'y': 2}
```



# Método *clear* (e *len*)

- `clear()`
  - Remove todos os elementos do dicionário
  - Ex.:

```
>>> x = { "Joao": "a", "Maria": "b" }
>>> x.clear()
>>> print x
{}
```
- Observação: Assim como em lista, é possível saber quantos elementos existe no dicionário usando LEN
  - `len(x)`



# Método *update*

- `update(dic)`
  - Atualiza um dicionário com os elementos de outro
  - Os itens em *dic* são adicionados um a um ao dicionário original
  - É possível usar a mesma sintaxe da função `dict` para especificar *dic*
  - Ex.:

```
>>> x = {"a":1,"b":2,"c":3}
>>> y = {"z":9,"b":7}
>>> x.update(y)
>>> x
{'a': 1, 'c': 3, 'b': 7, 'z': 9}
>>> x.update({'r':3})
>>> x
{'a': 1, 'c': 3, 'b': 7, 'r': 3, 'z': 9}
```



# Exercício

- Uma estação meteorológica registra a temperatura ocorrida em uma cidade a cada hora. Faça um programa que:
  - Leia as temperaturas de uma semana no seguinte formato: dia, hora, temperatura. A leitura termina com um dia inválido.
    - Atenção, use um dicionário cuja chave são os dias da semana. Cada dia da semana deve ser mapeado para um outro dicionário mapeando horas em temperaturas.
  - Em seguida, o programa imprime a maior temperatura para cada dia da semana.
  - Finalmente, o programa deve então informar para cada hora consultada, a maior temperatura ocorrida na semana. As consultas terminam com um valor de hora inválido.



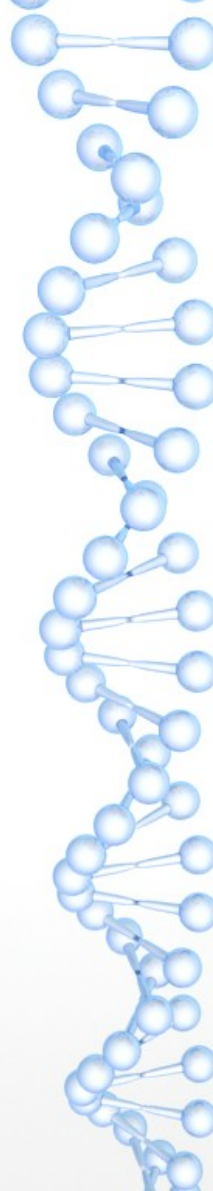
# Referencias

- Rodrigo de Toledo
  - [www.dcc.ufrj.br/~rtoledo/python/](http://www.dcc.ufrj.br/~rtoledo/python/)
  - Exemplos: <http://www.dcc.ufrj.br/~rtoledo/python/py/>



# Próxima Aula

- Python Web
  - Padronização de Projetos WEB
  - Preparo do Ambiente



*“Só existem dois dias no ano que nada pode ser feito.  
Um se chama ontem e o outro se chama amanhã”*

Dalai Lama