Lógica de Programação com Python

por Nilo Menezes

http://www.nilo.pro.br/python/

Python 2.7 – requer PyGame
Atualizada em 30/10/2012

Objetivos

- Introduzir os conceitos de programação
- Apresentar as principais estruturas de dados
- Construir programas exemplo
- Utilizar uma linguagem de programação (Python)

Você

- Apresentação
  - Nome
  - Curso
  - O que já sabe?
  - O que espera do curso?
  - Aprender lógica para que?

Por que programar é difícil?

O que fazer ?

Por que Python?

Exercício 1

- Faça uma descrição em português de como desligar o computador.

Uma abordagem gráfica

- A programação de computadores é intimamente relacionada com a matemática e diversos problemas computacionais.
- Uma abordagem gráfica é sugerida para facilitar a introdução aos principais conceitos.

Uma abordagem gráfica

- Para desenhar, precisamos definir um sistema de coordenadas para corretamente identificar os pontos de um desenho.
- Sistema de coordenadas
- Utilizaremos daqui para frente coluna e linha para indicar uma posição.
- 5 x 10 significa: coluna 5, linha 10

Uma abordagem gráfica

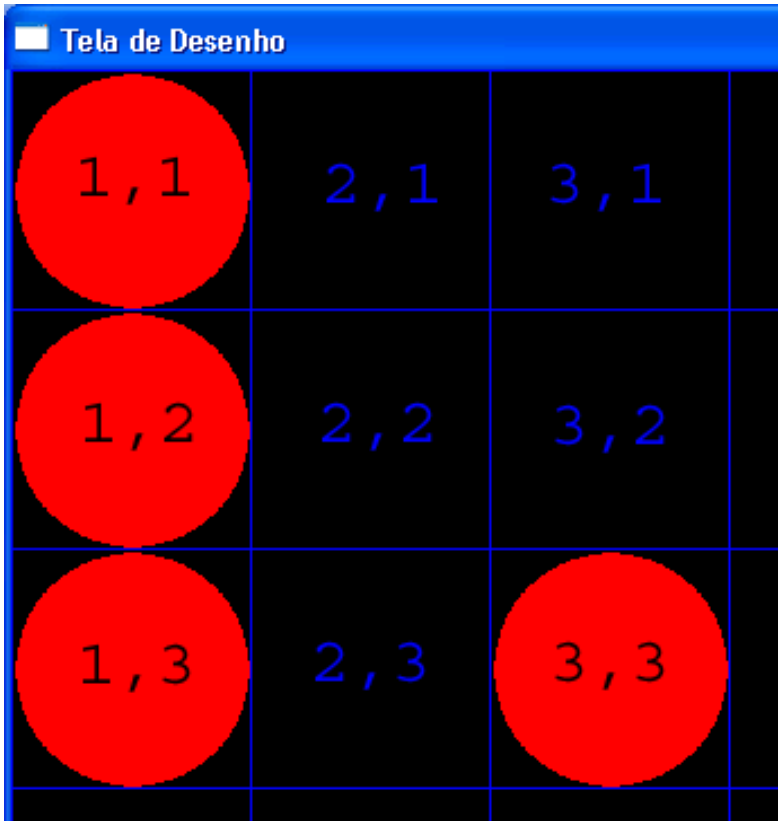
- Um ponto

  ponto(coluna, linha)
- Três pontos

  ponto(1,1)

  ponto(1,2)

  ponto(1,3)



Exercício 2

- Faça um programa para desenhar uma linha vertical numa tela de 5x5

Exercício 3

- Faça um programa para desenhar uma linha horizontal numa tela de 5x5

Exercício 4

- Faça um programa para desenhar uma linha diagonal numa tela de 5x5

Exercício 5

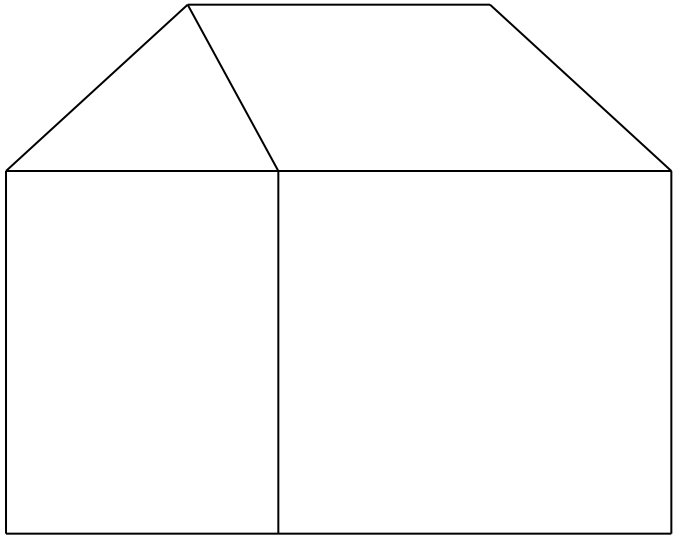
- Faça um programa para desenhar um triângulo

Exercício 6

- Faça um programa para desenhar um quadrado

Exercício 7

- Faça um programa para desenhar uma casa em 20x20



Tipos de Dados

- Uma das principais funções de um programa é a manipulação de dados.
- Para entendermos corretamente como fazê-lo, precisamos entender os tipos de dados e suas diferenças.

Tipos de Dados

- Número
  - Inteiros (1, 2, 3...)
  - Ponto Flutuante (1.14, 3.1415, 5.0)

Tipo Literal

- Composto por letras e números
- Escrito entre aspas

A = “texto”

- Não confundir A com “A”

A é a variável A
“A” é o literal A

Tipos de Dados

- Lógico
  - Verdadeiro
  - Falso

Estes valores também são chamados de booleanos.

Operadores Aritméticos

+ Adição

- Subtração

* Multiplicação

/ Divisão

- Em expressões mantém-se a prioridade das operações como na matemática.
- Parênteses () podem ser usados para alterar esta ordem.

Operadores Lógicos

Não

E

Ou

- Operadores lógicos são utilizados para modificar valores como verdadeiro e falso, criando expressões lógicas.
- O resultado das operações é definido pelas chamadas tabelas-verdade de cada operador

Operador Não

Valor	Não Valor
Verdadeiro	Falso
Falso	Verdadeiro

Operador E

A	B	A e B
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

Operador Ou

A	B	A ou B
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

Prioridade de Avaliação

- Quando mais de um operador fizer parte de uma sentença, é necessário seguirmos a seguinte ordem: não, e, ou.
- Para operadores aritméticos, utiliza-se o padrão da matemática:
  1) Multiplicação e Divisão
  2) Adição e Subtração

Exercício 8

- Sendo A verdadeiro e B falso, resolva:
  a) A e B
  b) B e não A
  c) A ou B
  d) A e B ou não B
  e) não B

Operadores Relacionais

- == Igualdade
- > Maior que
- < Menor que
- >= Maior ou igual
- <= Menor ou igual
- <> Diferente de

Prioridade de Operações

- Havendo vários tipos de operadores em um expressão, executam-se nesta ordem:
  1. Aritméticos
  2. Relacionais
  3. Lógicos

Exercício 9

- Resolva:
  a) 5 * 4 < 4 + 3
  b) 6 * 2 – 1 > 3 * 1
  c) 9 – 4 / 2 <= 7 + 1 ou 5 * 2 – 3 <> 6
  d) 9 / 3 == 3 * 3 e 2 * 3 – 1 >= 8

Variáveis

- São nomes usados para representar valores
- Possuem um tipo de dados
- Só podem armazenar um valor de cada vez
- Devem ter nomes começando com letras ou _. Podem conter números, exceto no primeiro caractere

Atribuição

- Variável = expressão
- Exemplo:

  A = verdadeiro

  B = 5 * 3

  C = A e B

  D = B * A – 2 > 4
- A atribuição é a operação que modifica o valor de uma variável.
- É importante notar que se atribui o resultado da expressão a variável e não a expressão em si.

Seqüência

- Uma seqüência de instruções ou comandos se executa de cima para baixo.

  A = 4

  B = 5

  C = A + B

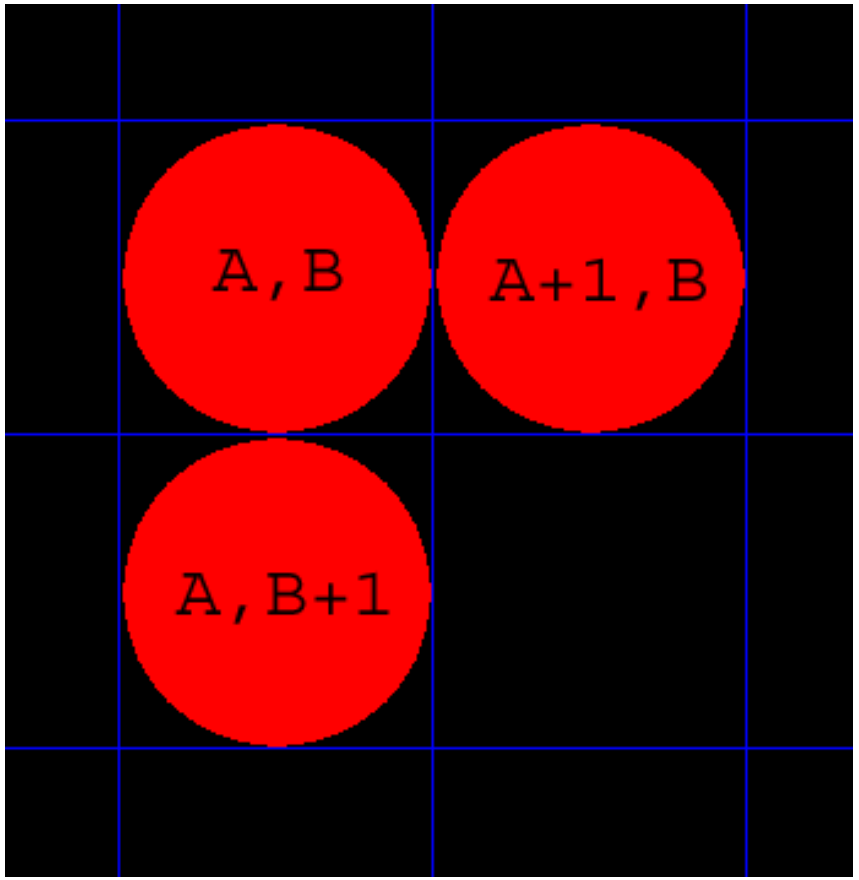
  A = 10

- No final, A vale 10, B vale 5 e C vale 9.

Observe que o valor de C não foi alterado pelo novo valor de A

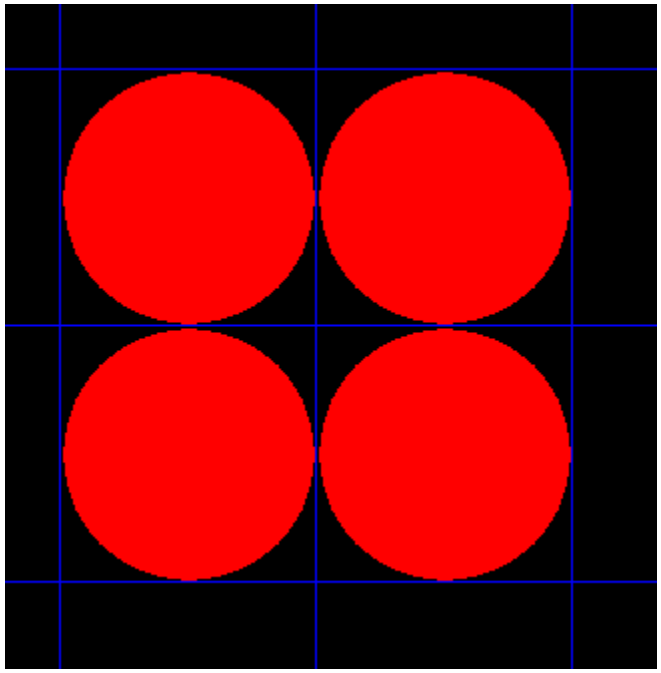
Coordenadas Relativas

- Podemos definir um desenho ou forma através de coordenadas relativas a um ponto. Imagine A,B como sendo as coordenadas de um ponto.
- O ponto ao lado é A+1,B
- O ponto abaixo é A, B+1



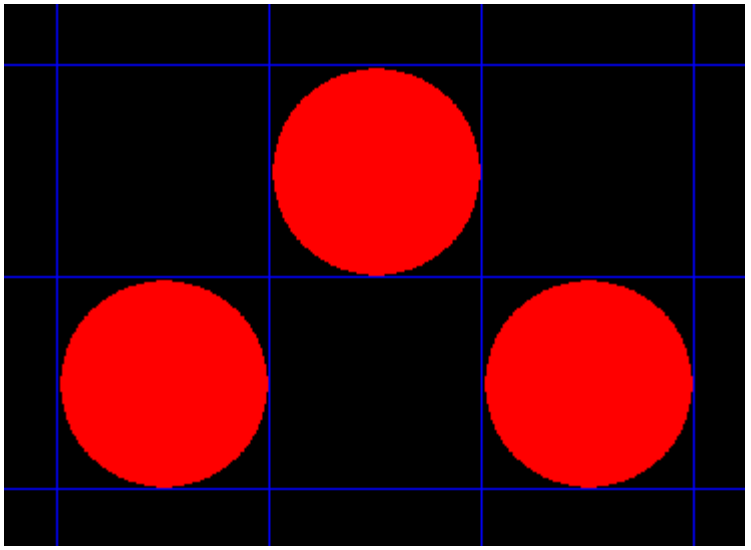
Exercício 10a

- Defina os quatro pontos que formam os cantos de um quadrado usando três variáveis.
- Uma para a linha, outra para a coluna superior esquerda.
- A terceira deve conter o tamanho do lado.



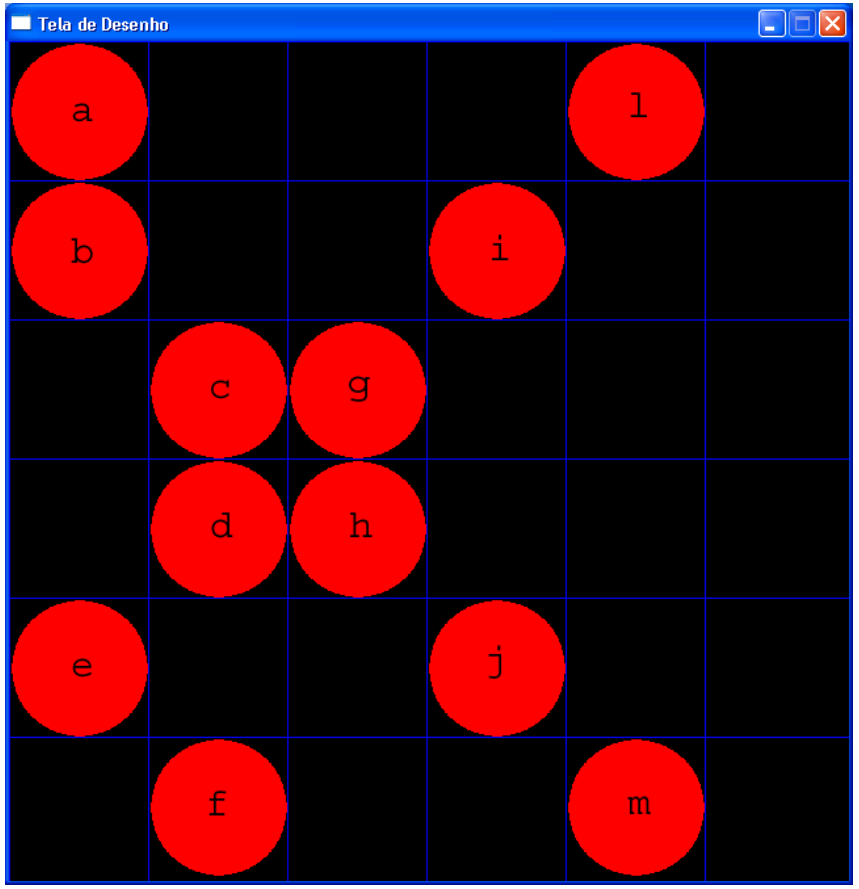
Exercício 10b

- Defina um triângulo de altura L.
- Defina-o usando A,B como o ponto da esquerda e depois faça para os outros dois pontos



Exercício 10c

- Escreva um programa para desenhar o gráfico ao lado.
- Utilize coordenadas relativas, considerando A,B o primeiro ponto.
- As letras indicam a questão e o ponto que deve ser usado como referência.



Saída

- Instrução que indica a exibição de informação, normalmente na tela

Escreva “Alô !!!”

- Utilizada também para exibir o conteúdo de variáveis e o resultado de expressões.

Escreva A

Escreva 2 * 5

Decisões

- Decidir o que fazer com base em um resultado lógico
- Consiste na escolha do que fazer, dependendo de uma condição
- Nem tudo segue uma ordem fixa
- Presente na maioria dos problemas

Decisões

- A decisão é composta de uma condição e um ou dois resultados. Um para o caso da condição ser verdadeira e outro caso falso.

Se a > b:

    Escreva "a é maior que b"

Senão:

    Escreva "a é menor ou igual a b"

Repetição

- Utilizada para delimitar um número de instruções ou comando que deve ser executado mais de uma vez.
- Utiliza uma condição para determinar se deve continuar a repetir (verdadeiro) ou não (falso).

Exemplo de repetição

```
A = 1
Enquanto A < 10:
    Escreva A
    A = A + 1
```

Entradas

- São pontos onde se pede a informação de um valor

Leia A

Exercício 11

- Escreva um programa que peça 2 números e exiba o maior deles.

Exercício 12

- Escreva um programa que pergunte um número e escreva a tabuada de multiplicar deste (1 até 10)
- Reescreva o programa anterior, desta vez perguntando o limite inferior e superior da tabela

Python e Lógica

- Fica mais difícil separar Python de Lógica
- Introdução ao IDLE
- Digitando programas

Expressões Lógicas

- Em Python, os operadores relacionais são iguais aos aprendidos em Lógica, exceto pelo fato de estarem escritos em inglês.
- Lembre-se sempre em minúsculas.

Lógica	Python
Não	not
E	and
Ou	or

Operadores Relacionais

> Maior	== Igualdade
< Menor	<> Diferente
<= Menor ou igual	!= Diferente
>= Maior ou igual	

Operadores Aritméticos

Operadores	
*	Multiplicação
/	Divisão
+	Adição
-	Subtração
**	Exponenciação

Escrevendo

print "Alô mundo!"

Imprime Alô mundo! na tela

print "O valor de %d x %d é %d" % (3,4,12)

Neste exemplo, %d é como uma lacuna onde preencheremos em ordem. O 3 para o primeiro, 4 para o segundo e 12 para o terceiro.

Escrevendo

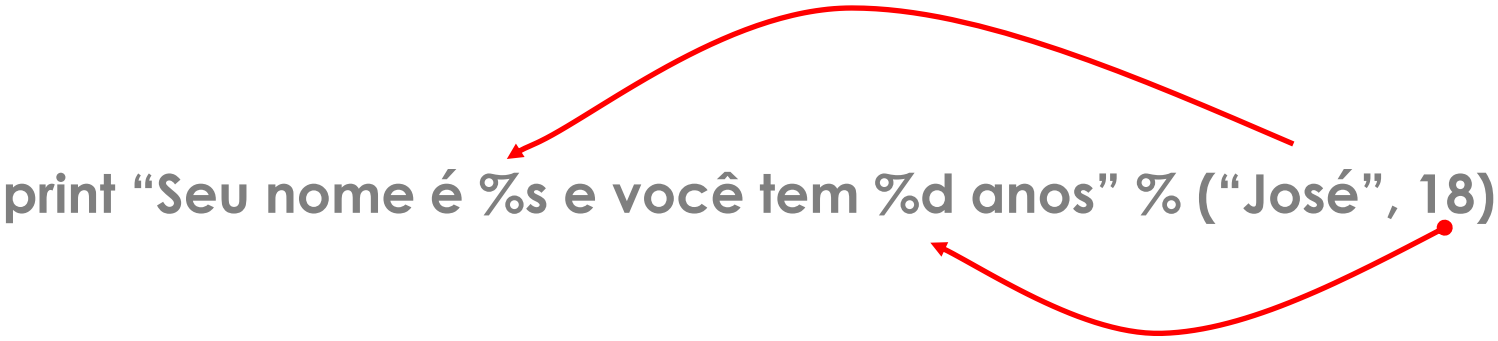
%d só funciona para valores inteiros

%s deve ser utilizado para valores literais

print "Seu nome é %s" % ("José")

Veja que o % aqui é usado apenas para separar a mensagem com máscara dos valores que serão utilizados entre parênteses.

Escrevendo



Seu nome é José e você tem 18 anos

Lendo

- A leitura em Python é especial:

C = raw_input(“Digite o valor de C”)

- Digite o valor de C será impresso antes de pedir a digitação do valor.
- raw_input sempre retorna um valor literal

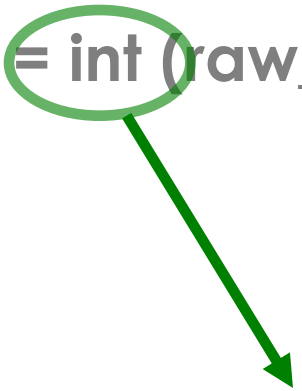
Lendo

Valores inteiros:

tamanho = input(“Digite o tamanho:”)

Ou

Tamanho = int (raw_input(“Digite o tamanho:”))



Conversão de tipo

Tipos

- Utiliza-se a função type para saber o tipo de uma expressão.

```
print type(1)
<type int>
print type(2.0)
<type 'float'>
print type("Alô")
<type 'str'>
```

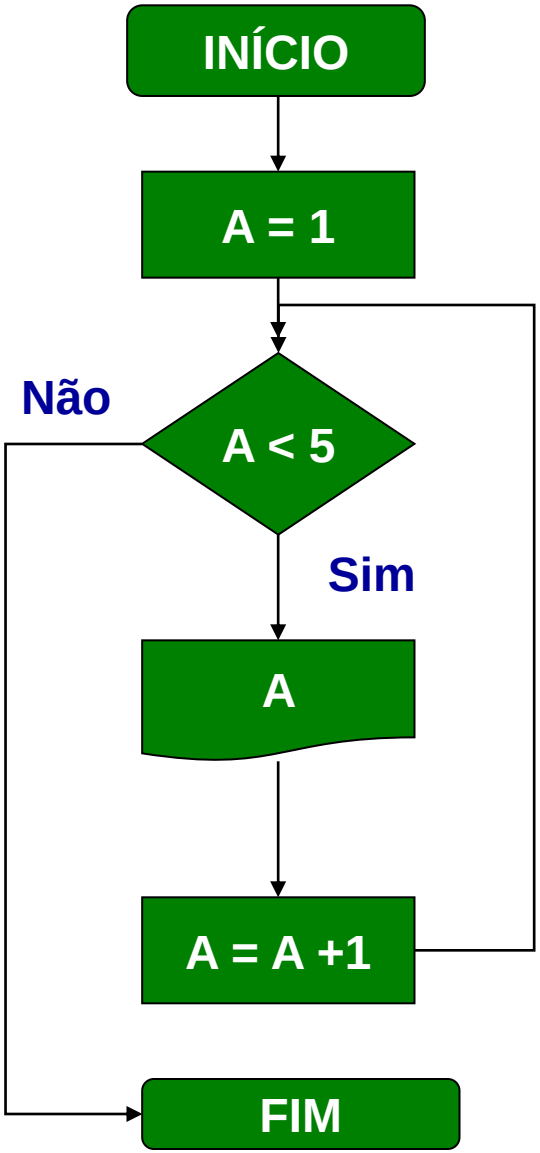
Repetição

Lógica	Python
A=1 enquanto A<5: escreva A A=A+1	A=1 while A<5: print A A=A+1

Repetição

- O bloco será repetido enquanto a condição for verdadeira.
- Após a última linha do bloco, a execução volta para a linha do while onde a condição será novamente avaliada.

Repetição



Definindo funções

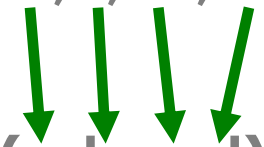
- Você pode definir ou criar suas próprias funções.

```
def quadrado(a,b,c,d):
    p = a
    while p<c:
        ponto(p,b)
        ponto(p,d)
        p=p+1
    p = b
    while p<d:
        ponto(a,p)
        ponto(c,p)
        p=p+1
```

Esta linha define o nome da nova função e quais parâmetros ela recebe

Definindo Funções

quadrado(10,8,20,15)



quadrado(a,b,c,d)

Seria o mesmo que:

a = 10

b = 8

c = 20

d = 15

Cores

- Em Python, utiliza-se RGB (Red, Green, Blue) para definirmos as cores.

(255, 0, 0)	Vermelho
(0, 255, 0)	Verde
(0, 0, 255)	Azul
(255,255,0)	Amarelo
(255,255,255)	Branco
(0,0,0)	Preto

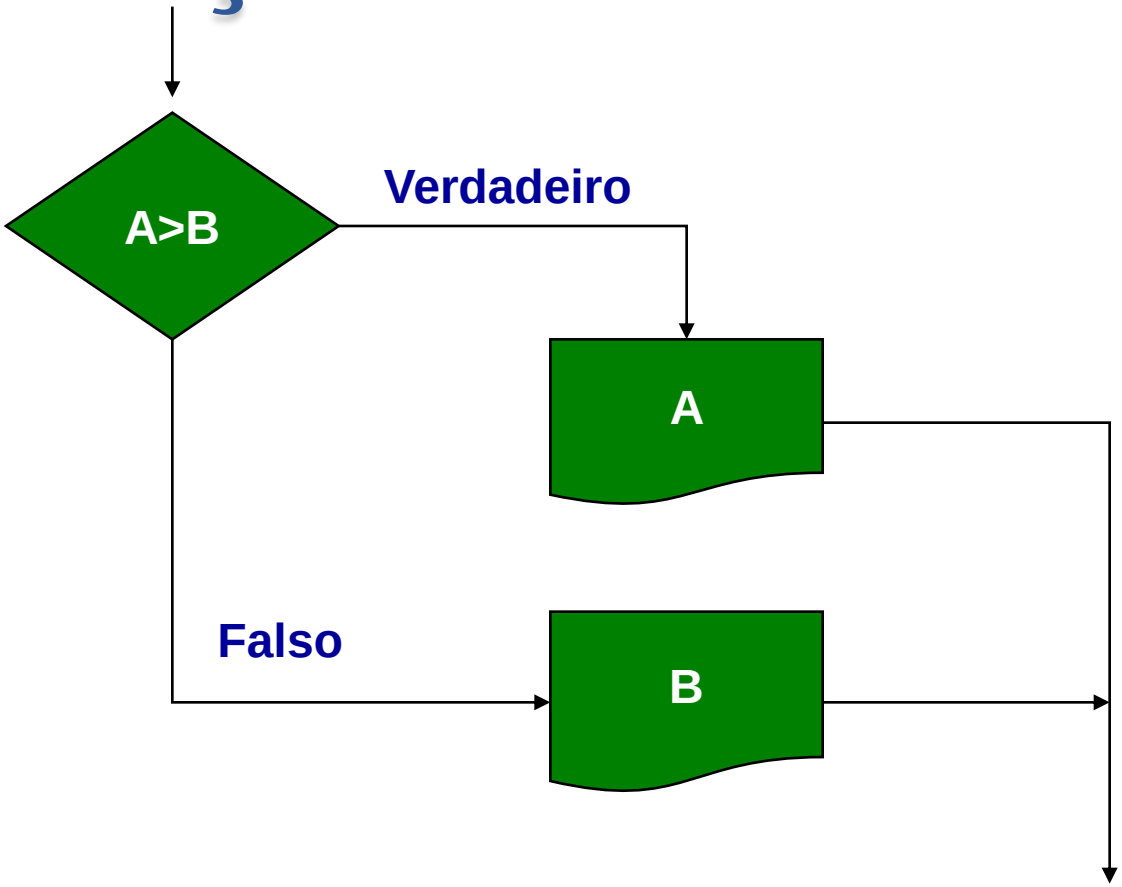
Condições

- O "se" da lógica vira if e o senão else.

Lógica	Python

se a>b:
    escreva a
senão:
    escreva b
	if a>b:
	    print a
	else:
	    print b

Condições



O utilitário de desenho

- Visite o site:
  - http://www.nilo.pro.br/python/
- Baixe os arquivos tela.py e graficos.py
- tela.py é o programa gráfico em si.
- Execute-o.
- Uma janela com grade deverá aparecer

graficos.py

- Este módulo possui uma série de funções que podem ser utilizadas em seus programas.
- Abra-o no IDLE.
- Execute com F5
- Digite: **ponto(1,1)**
- Um ponto vermelho deve aparecer na tela

graficos.py

- ponto(x,y)

  Exibe um ponto na cor atual na coluna x e linha y

- cor(r,g,b)

  Altera a cor atual para a definida por r,g,b

  (red = vermelho, green = verde, blue = azul)

graficos.py

- limpa()

  Limpa todos os pontos

- limpa(t)

  Limpa todos os pontos, redefinindo o gradeado para t x t

graficos.py

- inicializa()

  Estabelece a comunicação com a tela de desenho. Deve ser o dado antes de qualquer outro comando de desenho

- finaliza()

  Termina a comunicação com a tela de desenho

Usando o utilitário

- Usar a linha de comando é muito interessante no início, mas gravar seu desenho se torna cada vez mais importante.
- O utilitário gráfico pode ser utilizado em seus programas Python e não apenas pelo interpretador

Usando o utilitário

- Crie um novo programa em Python e digite:

```
from graficos import *

inicializa()
limpa(32)
cor(255,255,0)
ponto(1,1)
finaliza()
```

Exercício 12

- Crie uma função que desenhe a partir de uma coordenada (x,y):
    a) Uma árvore
    b) Uma casa
    c) Um sol

Exercício 13

- Faça um programa que desenhe uma paisagem com as funções do exercício anterior.
- Utilize mais de uma árvore e mais de uma casa em uma tela de 64x64 pontos.

Listas

- Listas são seqüências de valores
- Uma lista pode ter tamanho indeterminado
- Uma lista pode ser vazia
- Representa-se uma lista em Python através de colchetes

A = [1, 2, 3]

B = [ ]

Operações com Listas

- Os elementos de uma lista podem ser referênciados através de um índice, começando com 0 até o último elemento.

```
B = [1,2,3]
print B[0]
1
print B[2]
3
```

Operações com Listas

- **append(x)**

  Adiciona um item ao fim da lista.

```
B.append(5)
print B
[1,2,3,5]
```

Operações com Listas

- **len(L)**

  Imprime o número de elementos da lista L.

```
C=[10,40,50]
print len(c)
3
```

Operações com Listas

- **extend(L)**

  Concatena duas listas

```
A = [10, 11, 12]
B.extend(A)
print B
```

[1,2,3,5,10,11,12]

Operações com Listas

- **insert(x,l)**

  Insere um elemento l antes da posição x

print B
[1,2,3,5,10,11,12]
B.insert(3,4)
print b
[1,2,3,4,5,10,11,12]

Operações com Listas

- **remove(x)**

  Remove a primeira ocorrência de x na lista. Resulta em erro caso este não exista.

print B

[1,2,3,4,5,10,11,12]

B.remove(10)

print B

[1,2,3,4,5,11,12]

Operações com Listas

- **pop(i)**

  Remove o elemento i da lista. Se usado sem valor pop() retorna o último elemento da lista.

B = [1,2,3,4,5]

B.pop()

5

print B

[1,2,3,4]

Operações com Listas

- **index(x)**

  Retorna a posição do elemento cujo valor seja igual ao de x.

B = [1,2,3,4,5]

B.index(5)

4

Retorna um erro caso este não exista

Operações com Listas

- **count(x)**

  Retorna o número de vezes que x aparece na lista.

```
C = [1,1,2,2,3,3,3,3]
c.count(3)
4
```

Operações com Listas

- **sort()**

  Ordena os elementos da lista

```
D = [ 1,9,8,7]
D.sort()
print D
```

[1,7,8,9]

Operações com Listas

- **reverse()**

  Inverte a ordem dos elementos da lista.

```
C = [1,2,3,4,5]
C.reverse()
print C
```

[5,4,3,2,1]

Exercício 14

1. Crie uma lista com números de 1 à 10
2. Remova o elemento da posição 2
3. Remova o elemento de valor 5
4. Acrescente 11 e 12 ao fim da lista
5. Acrescente 0 no início da lista
6. Exiba o tamanho da lista

Exercício 15

- Escreva um programa para:
  - ler e imprimir 8 valores lidos do teclado.
  - Escrever a média e a soma destes valores.
  - Imprimir o menor e o maior.
  - Utilize um menu para:
    - 1. inserir valores. 2. Calcular média 3. Calcular soma 4. Calcular o maior 5. calcular o menor. Faça cada opção em um função diferente. Ao calcular, imprima os resultados.

Tipos variados

- Uma lista pode conter elementos de tipos diferentes.
- Exemplo:

F = [“joão”, 15, “maria”]

G = [10, 15, 20, True]

Listas em Listas

- Uma lista pode conter outra lista.

```
F = [ 1, 2, 3, [4,5,6]]
print F[3]
[4,5,6]
print len(F(3))
3
print len(F)
4
```

Listas em Listas

- Para referenciar um valor de uma lista dentro de outra lista, utiliza-se a mesma notação de listas com mais um índice.

```
F = [1,2,3,[4,5,6]]
print F[3][0]
```

4

Exercício 16

- Faça um programa que utilize listas para gerenciar uma agenda de telefones.
  - A agenda deve guardar nome e telefone de várias pessoas.
  - Operações a suportar: inclusão, exclusão, alteração, pesquisa, listagem e ordenação.
  - Em todos as opções o nome do indivíduo será utilizado como chave de pesquisa.
  - **Utilize menu**.

Funções com listas

- **range(tamanho)**

  Cria uma lista de 0 até o valor especificado em tamanho menos 1.

range(10)

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

Funções com listas

- **range(início, fim)**

  Cria uma lista iniciando no valor especificado em início e terminando em fim – 1

range(5,10)

[5,6,7,8,9]

Funções com listas

- **range(inicio, fim, incremento)**

  Igual à anterior, mas com incremento entre os elementos

range(1,10,2)

[1,3,5,7,9]

Funções com listas

- **xrange**

  Idêntica a todas as outras formas de range, porém otimizada para listas grandes.

for

- Estrutura de repetição que percorre os elementos de uma lista.

for elemento in lista:

- Exemplo: Imprimir todos os elementos

```
for a in range(10):
        print a
```

Usando Listas como pilhas

- Uma pilha é uma estrutura de dados com uma política de inclusão e exclusão de elementos bem definida.
- Em pilhas, a inclusão sempre é feita no topo ou fim, assim como as exclusões.
- A estrutura é similar a uma pilha de pratos.
- Coloca-se sempre um sobre o outro e retira-se sempre pelo topo.

Usando Listas como pilhas

```
pilha = [3, 4, 5]
pilha.append(6)
pilha.append(7)
print pilha
```

[3,4,5,6,7]

```
pilha.pop()
```

[3,4,5,6]

```
pilha.pop()
```

[3,4,5]

Usando listas como filas

- Filas são estruturas de dados onde se insere no fim e retira-se no início.
- É uma estrutura similar a uma fila de cinema.
- O primeiro que chega é o primeiro a entrar.
- Os que chegam depois vão para o fim da fila.

Usando listas como filas

Fila = [“João”, “Maria”]

Fila.append(“José”)

Fila.pop(0)

“João”

print Fila

[“Maria”, “José”]

filter

- Formato:

filter (função, lista)

- A função filter aplica a função passada como parâmetro a cada elemento da lista, retornando uma outra lista onde com os elementos cujo retorno de função é verdadeiro.
- Utilizada para filtrar ou selecionar valores.

filter

```
def f(x):
    return x % 2 == 0

print filter(f, [0,1,2,3,4,5,6,7,8])
```

[0,2,4,6,8]

map

- Formato:

map(função, lista)

- map funciona de forma semelhante a filter, porém esta retorna uma lista com o retorno de função aplicada a cada elemento da lista.
- Mais de uma lista pode ser passada como parâmetro. Neste caso, o número de argumentos de função deve ser igual ao de listas.

map

```
def fatorial(x):
        if x <=1:
                return 1
        else:
                return x * fatorial(x-1)
map(fatorial, [1,2,3,4,5])
```

[1, 2, 6, 24, 120]

map

- Um caso especial de map é passar None como função.
- Neste caso, map retornará uma lista com os argumentos que seriam passados.

map(None, [1,2,3],[4,5,6])

[(1,4), (2,5), (3,6)]

reduce

- Formato:

  reduce(função, lista)

- Aplica função aos elementos da lista.
- Na primeira vez, passa o primeiro e o segundo elemento.
- Nas próximas, o resultado da chamada anterior com o próximo elemento.

reduce

```
def mult(x,y):
        return x * y
reduce(mult, [1,2,3,4,5])
```

120

- Um terceiro parâmetro pode ser passado, indicando o valor inicial.

del

- Del é utilizada para remover um elemento de uma lista através de seu índice.

del lista(x)

- Remove o elemento x de lista

A = [1,2,3]

del a[0]

print A

[2,3]

- Del também pode ser utilizada para apagar variáveis.

Tuplas

- Tuplas funcionam como e compartilham muitas das propriedades de listas, porém tuplas não podem ser alteradas.
- Utilizamos ( ) invés de [] para denotar tuplas.

T = (1,2,3)

J = (“joão”, “maria”, “josé”)

Tuplas

- Uma tupla vazia é criada por:

T = ()

- Uma tupla com um elemento exige vírgula após este:

T = (“joão”,)

Tuplas

- Tuplas podem ser usadas para atribuições múltiplas.
- Exemplo:

A, B = (10, 15)

- É equivalente a A = 10 e B = 15

Strings

- Strings podem ser manipuladas como tuplas.
- E assim como tuplas não podem ser alteradas.

Nome = “João”

print Nome[0]

J

Strings

Para alterar uma String pode-se convertê-la em lista.

```
S = "João"
SL = list(S)
print SL
['J','o','ã','o']
```

Strings

- Para converter uma lista em string, utilize o método join da string.

S = S.join(SL)

- Embora não possamos alterar strings, nada nos impede de criarmos novas.
- S = S + "A"

Fatias

- Tuplas, listas e strings suportam um tipo de operação chamado slicing (fatiamento).

A = [1,2,3,4,5]

print A[0:2]

[1,2]

print A[1:]

[2,3,4,5]

print A[:4]

[1, 2, 3, 4]

Fatias

- A notação de fatia é [inicio:fim] sendo início inclusive, mas fim não.
- Isto é, [1:4] inclui o elemento 1, mas não o elemento 4.
- [:] pode ser utilizado para criar uma nova cópia de string ou lista.
- Valores negativos indicam que a contagem se inicia do fim da seqüência.

Dicionários

- Dicionários são estruturas de dados que permitem indexar um elemento através de sua chave.
- Dicionários são escritos entre chaves e sempre no formato chave:valor.
- Exemplo:

Telefones = { “nilo”:9717, “joana”:9784}

Dicionários

- Para acessar os valores de um dicionário, colocamos a chave entre colchetes, com em listas:

print Telefones[“nilo”]

9717

Dicionários

- O método has_key pode ser utilizado para verificar se uma chave existe.

Telefones.has_key("nilo")

True

Telefones.has_key("maria")

False

- Você também pode obter o mesmo efeito com in:

"nilo" in Telefones

True

"maria" in Telefones

false

Dicionários

- Para adicionarmos elementos a um dicionário basta especificar uma nova chave.

Telefones[“maria”] = 9784

- para listar todas as chaves de um dicionário, utilizamos o método keys() que retorna uma lista.

print Telefones.keys()
[“nilo”, “joana”, “maria”]

Dicionários

- Utiliza-se del para remover um elemento de um dicionário.

del Telefones[“nilo”]

- Dicionários podem ser construídos através de uma lista de tuplas e do método dict.

print dict([(“nilo”, 9717),(“joana”, 9784)])

{ “nilo”: 9717, “joana”:9784 }

Dicionários

- Utiliza-se o método items para retornar uma lista com tuplas chave, valor.

print Telefones.items()

[(“nilo”, 9717), (“joana”, 9784)

- Utiliza-se o método values para retornar um lista com os valores.

print Telefones.values()

[9717, 9784]

Operações avançadas

- Criação de listas com notação de função (List comprehensions).

[x * 2 for x in range(5)]

- Cria uma lista [0,2,4,6,8]

Operações Avançadas - Lambda

- Funções lambda são semelhantes a funções, mas não tem nome e podem ser utilizadas como variável.

```
F = lambda x: x+5
print F(4)
```

9

Arquivos

- Arquivos são estruturas de dados armazenadas em disco.
- Em Python, utiliza-se a função open para abrir um arquivo.

```
F = open(nome_do_arquivo, modo)
print F.readline()
F.close()
```

Arquivos

- Modos para abertura de arquivos:

Modo	Função
r	Leitura
w	Escrita. Apaga caso já exista
a	Abre para adicionar ao final (append)
r+	Leitura e escrita
b	Indica modo binário

Arquivos

- Todo arquivo aberto deve ser fechado com uma chamada a close.

Arquivos

- **read(x)**

  Lê um deterninado número de bytes (x) retornando como string.

- **readline()**

  Lê e retorna uma linha.

- **readlines()**

  Retorna todas as linhas de um arquivo numa lista.

Arquivos

- **xreadlines()**

  Semelhante ao xrange, otimizada para ler arquivos muito grandes.

- **tell()**

  Retorna a posição corrente no arquivo

- **seek(posição, referência)**

  Move o ponteiro do arquivo para posição.
  Refência (0 início, 1 atual, 2 fim)

Exceções

- Muitas funções em Python fazem o programa terminar em caso de erro.
- Isto acontece porque uma exceção foi gerada.
- Exceções não tratadas causam o fim do programa.

```
try:
    programa
except:
    tratamento
```

Onde aprender mais

- Visite o site:

http://www.nilo.pro.br/iprog/

para uma versão atualizada e completa do curso de introdução à programação.