Universidade Federal de Pernambuco
Centro de Informática

Programa e Comandos Básicos

Prof. Roberto Souto Maior de Barros
roberto@cin.ufpe.br

Conceitos Básicos

- Programa: uma visão mais prática...
  - Entradas
  - Manipulação da memória (Processamento)
  - Saídas
- A parte principal do programa é a manipulação da memória.
- As entradas e saídas servem para o programa se comunicar com o mundo externo.

Conceitos Básicos

- Memória: como funciona?
  - Semelhante à memória de calculadoras
    - Operações básicas são as mesmas: só duas...
    - Porém, mais operações compostas...
  - Quantidade muito grande de posições/espaço
    - Como se fosse ilimitado/infinito, por causa do uso de memória secundária (em disco), se necessário...
    - Necessidade de nomes, semelhante ao caso de calculadoras mais sofisticadas...
      - Mas seria inadequado se precisássemos saber os nomes reais/físicos para poder usá-las...

Conceitos Básicos

- Variável
  - É uma *abstração* que permite que o programador use um *identificador* (nome) para se referir a um espaço de memória utilizado para armazenar valores.
  - O tamanho da área de memória reservada para uma variável normalmente depende do tipo de dados que se pretende armazenar.
  - Por isto, em muitas linguagens de programação, é preciso dizer o tipo da variável explicitamente.
    - Este *não é* o caso de Python.

Programa Básico Python

- Estrutura de um programa Python em geral...

```
# Observe que isto é apenas um comentário.
definiçõesFunções
comandosPrograma   # Aqui também pode...
```

- Estrutura Recomendada de um programa Python

```
# Este programa faz....
definiçõesFunções   # Isto é opcional
comandosCriaçãoVariáveis   # Eu recomendo!
comandosPrograma   # solução do problema.
```

Identificadores

- Nomes criados pelo programador. Idealmente:
  - Dão uma idéia de para que servem...
  - Não são nem muito grandes nem muito pequenos.
- Regras
  - Combinação de letras, números ou _
    - Preferencialmente usar só letras e números...
  - Não pode começar por números.
  - Não pode ser igual a nenhuma palavra reservada.
  - Letras maiúsculas e minúsculas são diferentes!
    - Não é boa prática usar nomes semelhantes...

Identificadores

- Exemplos de identificadores *válidos*
  - soma
  - Temp01
  - nomeLongoDeVariavel
  - fortuna_total
  - _numClientes
- Exemplos de identificadores *inválidos*
  - 10var
  - var 623
  - #x
  - .valor-Saldo

Palavras Reservadas

- Palavras reservadas (*keywords*)
  - Python tem poucas...

```
and  as  assert  break  class  continue  def
del  elif  else  except  finally  for  from
global  if  import  in  is  lambda  not  or
pass  print  raise  return  try  while  with
yield  exec  nonlocal
```

Observação: Várias outras palavras são nomes de tipos, constantes e funções que são comuns e devem ser evitadas, apesar de não estarem reservadas.

Exemplo: str, int, bool, True, False, sqrt, pi, trunc, etc.

Declaração de Variáveis

- Em Python, *não há* uma declaração explícita para a criação das variáveis:
  - A sua primeira utilização é que será a sua declaração (implícita) - tipicamente um comando de atribuição.
  - O tipo de dados usado na criação será o tipo do valor usado neste primeiro comando.

- Obs: recomendo fortemente que estas primeiras atribuições de todas as variáveis do programa sejam agrupadas no início do mesmo.
  - Isto facilita o entendimento e manutenção do programa.

Declaração de Variáveis

- Em Python, é possível mudar o tipo associado a uma variável:
  - Basta fazer a atribuição de um valor de tipo diferente...

  Obs: Isto deve ser evitado pois é considerado péssima prática de programação.

- Alguns tipos básicos de Python:
  - *int, long, float, string, list, tuple, dictionary, file.*

  Obs: Python não tem um tipo *booleano* verdadeiramente. O tipo bool e suas constantes na verdade também são inteiros, i.e. são só "açúcar sintático".

Comando de Atribuição

- Sintaxe Genérica

  variável = expressão

  Onde

  – variável é o nome de uma variável; e

  – expressão é uma expressão qualquer cujo resultado (idealmente) seja do *mesmo tipo* da variável

    - As expressões mais simples são as *constantes* e os *nomes de outras variáveis* do *mesmo tipo* de variável.
    - Se a expressão for de tipo diferente, Python mudará o tipo da variável, enquanto a maioria das linguagens considera que a expressão está errada.

Comando de Atribuição

- Exemplos

```
idade = 5
fatorial = 1L
fatorial = idade
tempCelsius = 38.5
tempFarenh = tempCelsius
exclama = '!' ; fruta = 'banana'
nomeAluno = "Maria José da silva"
v1 = v2 = v3 = 0   # Bom só na inicialização.
```

Troca de Valores entre Variáveis

- Suponha que existem duas variáveis v1 e v2 com valores armazenados. Um trecho de programa para trocar o conteúdo destas variáveis:
  - Ou seja, se v1 tiver o valor 5 e v2 o valor 10, o código fará v1 armazenar o valor 10 e v2 o valor 5...
- Na maioria das linguagens seria implementado por algo como:
  - aux = v1 ; v1 = v2 ; v2 = aux
- Obs: Lembrem-se que variáveis se referem a posições de memória...

Atribuição Paralela

- Estrutura de Python que não existe em outras linguagens. Sua sintaxe genérica é:

  var1, var2, ... = expr1, expr2, ...

  Significado Filosófico:

  – execução simultânea das atribuições das expressões às variáveis correspondentes.

  Exemplos:

```
v1, v2 = v2, v1   # Troca os valores de v1 e v2.
v1, v2, v3 = v2, v3, v1   # Semelhante com 3 variáveis.
a, b, c = 0, 1, 2   # Não precisavam ser paralelas...
```

Atribuição Paralela (cont.)

Significado Real:

– variáveis auxiliares temporárias são utilizadas para conseguir este efeito, mas o que ocorre de verdade é a execução da esquerda para a direita, depois que todas as expressões são calculadas e salvas...

Exemplo:

```
a, a, a = 1, 2, 3   # Resultado é...
```

Obs: Só usar este tipo de atribuição se for tirar alguma vantagem...

Entrada de Dados Básica

- Comando input:
  - Principal comando de leitura.
  - Na versão 3, tudo é lido *sempre* como *string*.
  - No caso de strings (versões 1 e 2), o usuário precisa digitar os dados entre aspas (“) ou apóstrofos (‘).
  - Exemplo: var = input (“Mensagem qualquer: ”)
- Comando raw_input (equiv. ao input da versão 3):
  - Lê tudo como string: não precisa digitar delimitadores.
  - Exemplo: var = raw_input (“Mensagem qualquer \n”)

  Obs: O \n serve para mudar de linha no prompt...

Saída de Dados Básica

- Comando print:
  - Escreve na saída o conteúdo que segue.
  - Conteúdos múltiplos são separados por vírgulas e são normalmente impressos com um espaço de separação.
  - Cada *print* começa uma linha exceto se o último *print* tiver mudado isso no final. No seu conteúdo:
    - Nome de variável escreve seu conteúdo.
    - Constantes String precisam ser delimitadas por “aspas” ou ‘apóstrofos’. Dentro delas pode-se usar também:
      - \n para mudar de linha na impressão.
      - \t para dar um *<tab>* na impressão.
      - \”, \’ e \\ para escrever o símbolo após a \.

Saída de Dados Básica

- Exemplos de comando print:

```
print (a)
print (‘Resultado é’, a)
print (‘A =’, a, end=‘’)   # ou  print ‘A =’, a,
print (‘e B =’, b)
print (‘A =’, a, ‘\t B =’, b)
print (‘A =’, a, ‘\n B =’, b)
```

- Obs: No Python 3, o print é uma função e precisa que o conteúdo seja escrito entre parênteses. No Python 2 os parênteses não são usados.

Primeiro Programa Python

```
# Este programa repete a idade digitada.
idade = 0   # Declara a variável idade...
idade  = int(input("Digite a sua idade: "))
print  ("A idade digitada foi", idade)
```

Constantes dos Tipos Primitivos

char	‘A’, ‘\u0000’ a ‘\uFFFF’ (Hexadecimal)
int	Sem limites (Python 3)
int	-2147483648 a 2147483647 (Python 2)
long	-9223372036854775808L (Python 2)
float	5.0, +5.2, -4e-32, 1.8E308

Obs: A função type pode ser usada em variáveis e retorna o tipo atual do valor armazenado.

Exemplo: type (nome)

Expressões Aritméticas

- Operadores Aritméticos Básicos: + -, *, /, //, **
  - Lembrar que em Python 2 a divisão (/) entre números inteiros terá resultado inteiro (igual a //)!
    - 7 // 2 e 7 / 2 são 3, mas 7.0 / 2 é 3.5.
- Prioridade igual à da matemática...
  - Se as prioridades forem iguais, o cálculo é feito da *esquerda para a direita*
  - Parênteses para mudar a prioridade
  - Exemplo de expressão válida: -(2 + 3) * 4.5 / (5 - 2)
- Resto de divisão inteira (*mod*): %
  - 7 % 2 é 1

Expressões Aritméticas

- Nomes de variáveis podem ser usados nas expressões e implicam usar seu *conteúdo*...
  - *0.11 * saldo* também é uma expressão válida!
- Algumas funções nativas de Python:
  - int (x): transforma x em inteiro (se possível).
  - float (x): transforma x em tipo float.
  - str (x): transforma x em string.
  - pow (x,y): mesmo que x**y.
  - round (x, n): arredonda x para n casas decimais.
    - Se n for 0, resultado será valor inteiro mas tipo será float.

Expressões Aritméticas

- Não existe operador básico para *raiz quadrada.*
  - Mas biblioteca math disponibiliza esta e outras funções.
  - Antes, é necessário importar a biblioteca.
    - import math # Dá acesso às funções matemáticas.
  - Alguns exemplos:
    - math.sqrt (x) # significa raiz quadrada de *x.*
    - math.pi # Retorna a constante Pi.
    - math.trunc (x) # float -> int desprezando a parte decimal.
    - math.floor (x) # Mesmo resultado mas com tipo float.
    - math.ceil (x) # Semelhante mas valor vai para cima.
    - math.sin (x) # Retorna o seno de x.

Expressões Aritméticas

- Operadores Aritméticos com Atribuição
  - +=, -=, *=, /=
    - var1 += var2 significa var1 = var1 + var2
- Conversões entre Tipos Básicos em expressões
  - Podem ser necessárias em atribuições e também em outras situações semelhantes.
  - Em Python estas conversões são automáticas e podem mudar o tipo do resultado bem como o tipo da variável que recebe o resultado...
  - Logo, muito cuidado...

Fluxo dos Programas

- O fluxo da execução de programas escritos em linguagens de programação imperativas têm como *regra geral* seguir a ordem de escrita dos comandos.

- Porém, existem alguns comandos que mudam esta ordem natural de execução.
  - Exemplos: *if*, *for*, *while*, *etc.*

Comando *if*

- Sintaxe da versão mais simples

  if condição :

  comando

  Onde

  - condição é uma expressão *booleana*, ou seja, que retorna True ou False como resultado; e
  - comando é um comando válido qualquer.

  Significado:

  - Se condição for True executa o comando, senão não faz nada. Depois segue o fluxo normal de execução.

Comando *if*

- Sintaxe da versão completa

```
if condição :
    comando1
else :
    comando2
```

Significado:

– Se condição for True, executa o comando1, senão executa o comando2. Depois, segue o fluxo normal de execução.

Condição

- Em geral utiliza os *operadores relacionais* para formar expressões
  - ==, >, <, >=, <=, != (também <> em Python 2)
- Usa-se os operadores lógicos para formar condições compostas, mais elaboradas.
  - "E lógico" and
    - Só retorna True se os 2 operandos forem True.
  - "Ou lógico" or
    - Só retorna False se os 2 operandos forem False.
  - "Negação" not
    - Ou seja, not True é False e not False é True.

Condição

- O segundo operando de um and (e de um or) só será avaliado *se for necessário*.
- Pode-se usar parênteses para determinar a ordem de avaliação das sub-expressões em condições mais complexas.
- Variáveis do tipo bool *não* precisam usar os operadores relacionais:
  - Escreve-se *varBool* ao invés de *varBool == True*
  - Escreve-se *not varBool* para *varBool == False*

Exemplos de Comando *if*

```
if quant != 2 :     # Exemplo de if sem else
    quant = quant + 1
if ((v1 >= v2) and (v2 == v3))  or  varBool :
    quant = quant + v1 * 2
else:
    quant = v2 - v3
if  valor < 0 :
    total = total + valor
else:  # Os comandos dentro de um if ou else podem ser outro if
    if  valor > 10 :
        total = total + 2 * valor
    else:
        total = total + 3 * valor
```

Comando *if*

- Sintaxe específica de Python:

```
if  condição1 :
      comando1
elif condição2 :   # Isto é só uma abreviação
      comando2     # de else:  if...
elif ... :         # Impacto na indentação…
      ...
else:
      comandoN
```

Bloco de Comandos

- Na maioria das linguagens, serve para tornar possível escrever *mais de um comando* nos lugares em que só *um comando* é esperado:
  - em geral *dentro de outros comandos, p*or exemplo, *dentro* de um comando *if*.
- Em Python, blocos de comandos são implícitos, é só indentar os vários comandos para delimitar o bloco, i.e. é a indentação que define o contexto...
  - Em outras linguagens é necessário marcar o bloco explicitamente com chaves ({ e } - caso de Java e C) ou com as palavras begin e end (caso de Pascal).

Exemplos de *if* com blocos

```
if quant != 2 :   # Exemplo de if sem else
    quant = quant + 1
    valor = valor *10

if valor < 0 :
    total = total + valor
    valor = valor + 1
elif valor > 10 :   # Exemplo com elif
     total = total + 2 * valor
else:
     total = total + 3 * valor
     valor = valor + 5
```

Comandos de Repetição

- São comandos que servem para controlar a execução repetida de outro(s) comando(s)
  - Em Python: *for* e *while*.
- Na prática, em geral são vários os comandos a serem repetidos. Para isto usa-se um bloco de comandos:
  - Python: comandos a serem repetidos são indentados.
  - Outras linguagens: comandos são delimitados por { e } ou outros delimitadores.

Comando *for*

- Principais características (filosoficamente)
  - Repetição é controlada pelos valores de uma variável de controle sobre um intervalo numérico definido.
  - Número de repetições é portanto previsível!

- Observação importante
  - A sintaxe do comando *for* de Python é *diferente* da maioria das outras linguagens, já que é baseada em listas – o que veremos agora é apenas uma versão simplificada, que é mais parecida com o comando *for* das outras linguagens.

Comando *for*

- Sintaxe – versão simplificada

  for varCont in range (limIni, limFin, passo):

  comando(s)

  Onde

  - varCont é a variável que irá controlar a repetição do(s) comando(s) que forem escritos dentro do for.
  - limIni é o limite inicial do intervalo (fechado).
  - limFin é o limite final do intervalo (aberto).
  - passo (opcional) é o incremento que, a cada repetição do corpo do for, muda o valor da variável de controle – desde que fique dentro do intervalo [limIni..limFin).

Comando *for* – versão simplificada

for varCont in range (limIni, limFin, passo):

    comando(s)

Significado:

- A variável de controle varCont assumirá todos os valores possíveis dentro do intervalo [limIni..limFin).
- Para cada valor de varCont, o for executará os comandos e depois tentará fazer varCont = varCont + passo.
- Quando terminar, segue o fluxo normal de execução.
- Se não for informado, o passo assume o valor 1. Neste caso, limIni pode também ser omitido e terá valor 0.

Obs: Note que o for pode não executar nenhuma vez...

Comando *for* – versão simplificada

for varCont in range (limIni, limFin, passo):

    comando(s)

Outras observações:

- Em Python, após a execução completa do comando for, a variável de controle varCont estará com o *último valor assumido* dentro do intervalo [limIni..limFin).
  - Caso o comando for *não execute nenhuma vez*, varCont continuará com seu valor anterior ao comando for.
- Nas outras linguagens, varCont ficará com o primeiro valor após o intervalo considerado, e isto independe do comando for ter ou não executado alguma vez.
- Na maioria das outras linguagens, o limite final do intervalo também é fechado – i.e. ele ainda executa.

Exemplos de Comando *for*

```
# Primeiro exemplo – soma números de 1 a 50.
# Lembre-se que o final do intervalo é aberto...
soma = 0
for  i  in  range (1, 51):   # 51 é 50 + 1 e passo é 1.
    soma = soma + i       # No final i estará com 50.

# Outro exemplo – multiplicação + ordem decrescente
# + supondo que v1 e v2 já foram inicializadas...
prod = 1   # Elemento neutro na multiplicação é 1.
for  j  in  range (v1, v2 - 1, -2):
    prod = prod * j
```

Comando *while*

- Principais características
  - Repetição é controlada por uma condição qualquer.
  - Número de repetições é portanto imprevisível!

Sintaxe

```
while  condição :
        comando(s)
```

Onde

- condição e comando(s) têm o mesmo significado visto anteriormente – no comando if.

Comando *while*

while condição :

comando(s)

Significado:

- Se condição for *True* executa comando(s) e depois volta a testar condição.
- Se/quando condição for *False*, segue o fluxo normal de execução.
- Note que os comandos *podem não executar nenhuma vez...*

Observação Importante

- comando(s) deve de alguma forma permitir que condição passe a ser *False* após um número qualquer de passos.

Exemplos de Comando *while*

```
# Mesmo primeiro exemplo visto anteriormente com for.
soma = 0
i = 1
while  i <= 50 :   # Note que aqui i estará com 51 após terminar...
        soma = soma + i
        i = i + 1

# Exemplo sem um número previsível de repetições.
soma = 0
vInt = int(input("Digite um número inteiro (0 para parar): "))
while  vInt != 0 :
        soma = soma + vInt
        vInt = int(input("Digite outro número inteiro (0 para parar): "))
```

Leitura Múltipla de Dados

- Os comandos de repetição podem ser usados para ler múltiplos dados de um mesmo tipo.
- Basicamente existem dois padrões simples para leitura de múltiplos dados:
  1. Usuário informa *previamente* a quantidade de dados que será digitada – nestes casos, o comando *for* deve ser usado para controlar a leitura dos dados.
  2. Usuário informa os dados *livremente*, mas adota uma *convenção* qualquer para significar o *fim da digitação dos dados* – nestes casos, o controle da leitura deve ser feito com um comando *while*.

Leitura Múltipla de Dados

- Padrão 1 – “esqueleto”

```
# Usuário diz antes quantos dados digitará...
qtd = int(input(“Digite a quantidade de dados: ”))
for  i  in  range (1, qtd +1):   # Ou range (qtd)
    Leitura de uma ocorrência do(s) dado(s)
    Processamento do(s) dado(s) lido(s)
```

Obs: Observe que range (qtd) executa as mesmas qtd vezes, mas os valores de i serão diferentes.

Leitura Múltipla com *for*

```
# Exemplo – soma números digitados pelo usuário
# usando o padrão 1 de leitura.
i = num = soma = 0
qtd = int(input("Digite a quantidade de números: "))
for  i  in range (qtd):   # Intervalo 0 a qtd -1 será percorrido.
        num = int(input("Digite um número: "))
        soma = soma + num
print  ("A soma dos", qtd, "números digitados é", soma)
```

Leitura Múltipla de Dados

- Padrão 2 – “esqueleto”

Leitura de uma ocorrência do(s) dado(s)
while not Condição-de-parada :
    Processamento do(s) dado(s) lido(s)
    Leitura de uma ocorrência do(s) dado(s)

Leitura Múltipla com *while*

```
# Exemplo – soma números digitados pelo usuário
# usando o padrão 2 de leitura.
num = soma = qtd = 0
num = int(input("Digite um número (zero para parar): "))
while  num != 0 :   # Condição inversa à da parada.
        soma = soma + num
        qtd = qtd + 1
        num = int(input("Digite outro (zero para parar): "))
print  ("A soma dos", qtd, "números digitados é", soma)
```

Comandos a Evitar

- Comandos que “quebram” o fluxo normal de execução dos comandos de repetição
  - Só servem para que programadores *preguiçosos* escrevam código *mal escrito!*
- break
  - Força a saída do loop em comandos de repetição.
- continue
  - Força um novo teste da condição que controla o fim do loop sem terminar a execução atual do loop.
- else (se usado em for e while – só em Python)
  - Só faria/teria sentido se usasse break, logo não use...