Programação de Computadores

Instituto de Computação UFF
Departamento de Ciência da Computação

Otton Teixeira da Silveira Filho

Conteúdo

- Comandos de repetição
- Alguns aspectos no processamento numérico
- Atribuições múltiplas
- Exemplos
- Algumas funções implícitas de Python
- Aspectos dos tipos de dados no Python

Laços de repetição - for

Python tem um comando de repetição que apresentaremos na forma

```
for  elemento in iterador :
     comando(s)
```

OBS:

- Esta uma versão simples do for. **Ele tem mais recursos.**
- Um iterador é uma das particularidades de Python, embora exista algo equivalente em outras linguagens. Enquanto elemento estiver no iterador, a repetição será feita
- Aqui apresentaremos um criador de iterador que é muito útil: range()

Laços de repetição - for

Uma versão de uso do for:

```
for  contador in range(inicio, fim, passo) :
     comando(s)
```

Tanto o contador quanto o início da contagem, o fim da contagem e o passo da contagem serão int neste momento. range() permite maior amplitude de utilização como veremos em breve

- A omissão do passo de contagem suporá o uso de passo 1
- A contagem apresentada no range() será até fim - 1

Laços de repetição - for

Exemplo:

Imprima os números inteiros entre 1 e 10

```
# Imprime valores de 1 a 10

def main():

    for i in range(1, 11) :
        print(i)

main()
```

provocará a impressão dos números 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, um por linha.

Laços de repetição - for

Exemplo:

Imprima os números inteiros impares de entre 1 e 10

```
# Imprime valores impares entre 1 e 10

def main():

    for i in range(1, 10, 2) :
        print(i)

main()
```

provocará a impressão dos números 1, 3, 5, 7, 9, um por linha.

Laços de repetição - for

Some todos os números inteiros entre 1 e 100.

```
# Programa que soma os valores de 1 a 100

def main() :

    s = 0

    for i in range(1, 101) :
        s = s + i

    print("A soma dos numeros de 1 a 100 e' igual a ", s)

main()
```

Laços de repetição - for

Some todos os números inteiros entre 1 e 100.

```
# Programa que soma os valores de 1 a 100

def main() :

    s = 0

    for i in range(1, 101) :
        s = s + i

    print("A soma dos numeros de 1 a 100 e' igual a ", s)

main()
```

- Experimente mudar a indentação colocando o print() sob efeito do for

Laços de repetição - for

Imprima os números inteiros entre 1 e 10 regressivamente

```
# Imprime valores de 1 a 10 regressivamente

def main():

    for i in range(10, 0, -1) :
        print(i)

main()
```

provocará a impressão dos números 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, um por linha.

Laços de repetição - while

Usaremos um outro laço de repetição

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

o laço repetirá as operações dentro do bloco enquanto a condição lógica for verdadeira

- Podemos criar um “laço eterno“ se a condição lógica for sempre True

Laços de repetição - while

Some uma variável int a partir de 1 enquanto a soma seja menor que 1000. Imprima o valor do int para o qual a condição passou a não valer e também o resultado final.

```
# Programa usando while

def main() :

    s = 0
    i = 0

    while s < 1000 :
        i = i + 1

        s = s + i

    print(i, s)

main()
```

Laços de repetição - while

Faça um programa que solicite dois ints que deverão ser somados. O programa continuará solicitando dois int até que ambos sejam negativos.

Laços de repetição - while

```
# Programa usando while
# Exemplo de laco interrompido por uma condicao logica composta

def main() :

    print("Este programa solicitara' a entrada de dois int que serao somados.")
    print("So' parara' a solicitação se os dois numeros forem negativos")

    a = int(input("Entre com um numero: "))
    b = int(input("Entre com outro numero: "))

    while (a > 0) or (b > 0) :
        print("A soma de ", a, "e ", b, "e' igual a ", a + b)

        a = int(input("Entre com um numero: "))
        b = int(input("Entre com outro numero: "))

    print("Programa interrompido.")

main()
```

Laços de repetição - while

Observe a lógica do programa, incluindo o uso do or no contexto do while

Alguns fatos sobre computação numérica

Façamos um programa:

Atribua a uma variável float de identificador dx e conteúdo 0,01. Inicialize uma variável de acumulação com zero e acumule nela 100 vezes o valor da variável x. Imprima o resultado.

```
# Programa sobre tipos
# float nao e' um real
def main():

    dx = 0.01
    n = 100
    s = 0

    for i in range(0, n):
        s = s + dx

    print("O resultado de somar ", dx, n, " vezes e' ", s)

main()
```

Alguns fatos sobre computação numérica

Observe que **a soma de 0,01 cem vezes não deu 1!**

O que temos num computador são representações limitadas dos números da matemática, não são os números da matemática, sejam eles inteiros ou reais

Neste caso especial o resultado gerado é consequência de 0,01 ser dízima periódica em binário

Outro modo de usar o for

Mas poderíamos escrever este programa usando o for de modo mais sintético

```
# Programa sobre tipos
# float não e'um real
# for de um "parametro"
def main():

    dx = 0.01
    n = 100
    s = 0

    for i in range(n):
        s = s + dx

    print("O resultado de somar ", dx, n, " vezes e' ", s)

main()
```

Alguns fatos sobre computação numérica

Quando colocamos o for com apenas um elemento no range, a contagem partirá de zero

Alguns fatos sobre computação numérica

Mais um exemplo:

Faça 0,1 + 0,1 e teste se é igual a 0,2. Depois faça 0,1 + 0,1 + 0.1 e teste se é igual a 0,3.

```
# Programa sobre tipos
# float
def main():

    if(0.1 + 0.1) == 0.2 :
        print("0.1 + 0.1 e' igual a 0.2")
    else :
        print("0.1 + 0.1 nao e' igual a 0.2")

    if(0.1 + 0.1 + 0.1) == 0.3 :
        print("0.1 + 0.1 + 0.1 e' igual a 0.3")
    else :
        print("0.1 + 0.1 + 0.1 nao e' igual a 0.3")

main()
```

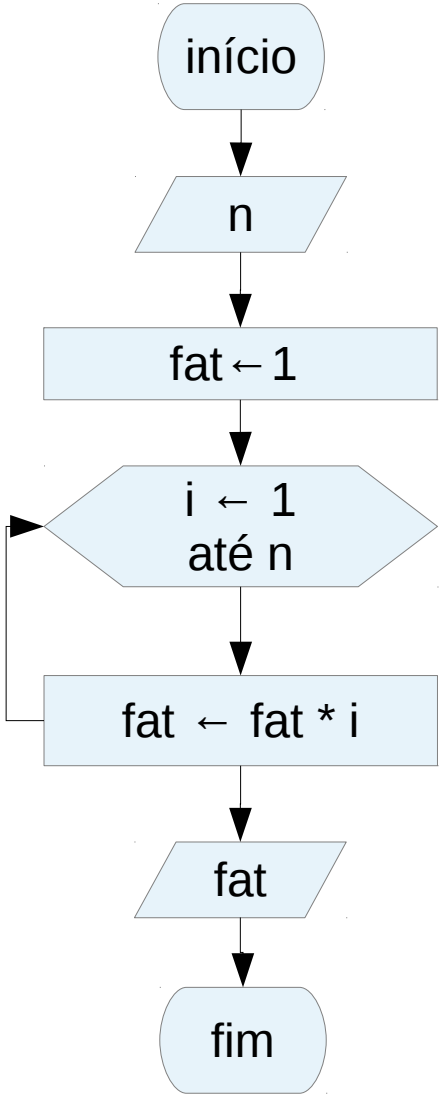
Outro exemplo

Novamente temos um resultado “estranho” se estivermos supondo que os números num computador são os números da matemática

Outro exemplo

Obtenha o fatorial de um número inteiro não negativo **n,** imprimindo o resultado calculado.

Outro exemplo




```
programa fatorial
 inteiro fat, i, n
 leia n
 fat ←1
 para i ← 1 até n
       fat ←fat * i
 fim para
 imprima fat
 fim
```

Outro exemplo

Fatorial de n

```
# Programa que calcula o fatorial de um numero inteiro dado

def main():

    n = int(input("Entre com um valor inteiro positivo : "))

    fat = 1

    for i in range(1, n + 1):
        fat = fat * i

    print("O fatorial de ", n, " e' ", fat)

main()
```

Outro exemplo

Calcule o valor da função exponencial no ponto **x=1** usando a série de Taylor truncada em **n** termos que é dada por

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Outro exemplo

```
Programa exponencial

real exp
inteiro fat, i, j, n

leia n

exp ← 1

para i ← 1 até n
    fat ←1
    para j ← 1 até i
        fat ←fat * j
    fim para

    exp ← exp + 1/fat
 fim para

 imprima exp

 fim
```

...

Outro exemplo

Programa exponencial

```
real exp
inteiro fat, i, j, n

leia n

exp ← 1

para i ← 1 até n
    fat ←1
    para j ← 1 até i
        fat ←fat * j
    fim para

    exp ← exp + 1/fat
 fim para

 imprima exp

 fim
```

```
# Exemplo de uso do for aninhado
# Programa que calcula e, base dos logaritmos neplerianos,
# com n termos da serie de Taylor

def main():
    n = int(input("Entre com um valor positivo "))

    exp = 1

    for i in range(1, n) :
        fat = 1

        for j in range(1, i + 1) :
            fat = fat * j

        exp = exp + 1/fat

    print("O valor procurado com", n, " termos e' ", exp)

main()
```

Laços de repetição

Se você achou um tanto confuso, não se preocupe

Esta implementação inclui o cálculo do fatorial com o da soma da série. É funcional mas não é algo recomendado.

Mais a frente veremos uma nova versão mais legível

Laços de repetição

Calcule o enésimo termo da sequência de Fibonacci

Laços de repetição

Calcule o enésimo termo da sequência de Fibonacci

A sequencia de Fibonacci é apresentada aqui da seguinte forma:

Dados como primeiros termos $F_1=1$ e $F_2=1$, geramos os próximos elementos usando a equação

$$F_n = F_{n-1}+F_{n-2}$$

Laços de repetição

Calcule o enésimo termo da sequência de Fibonacci

A sequencia de Fibonacci é apresentada aqui da seguinte forma:

Dados como primeiros termos $F_1=1$ e $F_2=1$, geramos os próximos elementos usando a equação

$$F_n = F_{n-1}+F_{n-2}$$

Então teríamos a sequência

1, 1, 2, 3, 5, 8, 13, 21, 34, 55...

Laços de repetição

Calcule o enésimo termo da sequência de Fibonacci

```
# Geracao do n-esimo termo da sequencia de Fibonacci
# Ha' um pequeno erro...

def main():

    a = 1
    b = 1

    n = int(input('Entre com n '))

    if n <= 0 :
        print('n negativo')

    elif n <= 2 :
        fibo = n
        print("N-esimo termo da sequencia de Fibonacci: ", fibo)
    else :
        while n > 2 :
            fibo = a + b
            a = b
            b = fibo
            n = n - 1

    print("N-esimo termo da sequencia de Fibonacci: ", fibo)

main()
```

Laços de repetição

O código anterior um tem erro

Veja se o localiza antes de avançar

Laços de repetição

Repare que se o valor dado for negativo, o if fará o fluxo do programa saltar para fora do encadeamento de if, onde fibo não está definido.

O interpretador de Python gerará um erro. Experimente…

Para corrigir, basta colocar o print() no nível de indentação do else

Laços de repetição

Calcule o enésimo termo da sequência de Fibonacci

```
# Geracao do n-esimo termo da sequencia de Fibonacci

def main():

    a = 1
    b = 1

    n = int(input('Entre com n '))

    if n <= 0 :
        print('n negativo')

    elif n <= 2 :
        fibo = n
        print("N-esimo termo da sequencia de Fibonacci: ", fibo)
    else :
        while n > 2 :
            fibo = a + b
            a = b
            b = fibo
            n = n - 1

        print("N-esimo termo da sequencia de Fibonacci: ", fibo)

main()
```

Atribuições múltiplas

Python permite que possamos fazer atribuições múltiplas, ou seja, fazer atribuições de valores simultaneamente e sequencialmente

Vejamos um exemplo numa modificação do programa anterior

Laços de repetição

Calcule o enésimo termo da sequência de Fibonacci

```
# Geracao do n-esimo termo da sequencia de Fibonacci
# Uso de atribuicao multipla em Python

def main():

    a = fibo = 1

    n = int(input('Entre com n '))

    if n <= 0 :
        print('n negativo')

    elif n <= 2 :
        fibo = n
        print("N-esimo termo da sequencia de Fibonacci: ", fibo)
    else :
        while n > 2 :
            a, fibo = fibo, a + fibo
            n = n - 1

        print("N-esimo termo da sequencia de Fibonacci: ", fibo)
main()
```

Laços de repetição

Observe:

ficou mais sintético mas de compreensão mais difícil no laço do while

Recordemos alguns do princípios que norteiam a linguagem Python

Zen do Python

- Belo é melhor que feio
- Explícito é melhor que implícito
- Simples é melhor que complexo
- Complexo é melhor que complicado
- Legibilidade conta

Tipos de variáveis

Programa **tipos.py**

Aqui constataremos mais uma sobrecarga e o cuidado que temos que ter ao especificarmos os conteúdos das variáveis

Executemos o programa que se segue.

- Sugestão: tecle dois “números” de um algarismo

Tipos de variáveis

Programa tipos.py

```
def main():
    a = input("Entre com a: ")
    b = input("Entre com b: ")

    print (a + b)
    print (a * 3)

    a = int(a)
    b = int(b)

    print (a + b)
    print (a * 3)
main()
```

Tipos de variáveis

Saida com programa tipos.py

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /home/otton/didatico/graduacao/prog/1.2018/Aulas/exercicios_Python/tipos.py
Entre com a: 3
Entre com b: 4
34
333
7
9
>>>
```

Tipos de variáveis

Programa **tipos.py**

Observe que input() sem a moldagem por int() dá como saída um tipo str, cadeia de caracteres. Antes de ser um algarismo, o símbolo que o representa é um caracter

- Vemos aqui que + é também um operador que concatena os caracteres de entrada, ou seja, não é aqui soma aritmética
- Aqui temos também o operador * repetindo o caracter um determinado número de vezes que foi especificado

Tipos de variáveis

Programa **tipos.py**

Aqui observamos novamente o que é chamado **sobrecarga de operadores**:

- os sinais + e * promovem resultados diferentes de quando operados não só com int e float como também com str!

Algumas funções

O Python contém funções pré-definidas

Algumas funções

O Python contém funções pré-definidas

Veremos mais tarde que o conceito de funções em linguagens de programação não correspondem totalmente ao que é chamado de função na matemática

Algumas funções

O Python contém funções pré-definidas

Veremos mais tarde que o conceito de funções em linguagens de programação não correspondem totalmente ao que é chamado de função na matemática

Aqui também seremos imprecisos não diferenciando funções de métodos, como também estamos fazendo com os conceitos de variáveis e objetos

Funções intrínsecas

Algumas da funções implícitas no Python

Nome	Definição	Argumento	saida
abs()	Valor absoluto	Int, float	Tipo de entrada
float()	Conversão para tipo float	número, cadeia de caracteres	float
int()	Conversão para tipo int	número, cadeia de caracteres	int
iter()	gera um iterador	“variado“	Um iterador
list()	Gera uma lista	“variado“	Uma lista
max()	***	“variado“	valor máximo do argumento
min()	***	“variado“	Valor mínimo do argumento
len()	***	“variado“	Tamanho do argumento
type()	***	“variado“	Devolve o tipo do argumento

Algumas funções

Repare que fomos informalmente apresentados a algumas funções como float(), int() e type()

Algumas funções

Este é um pequeno conjunto funções pré-definidas de Python

Veremos mais tarde os **módulos** que podemos carregar contendo funções mais específicas e abrangentes

Módulos

Três módulos importantes são:

- math, com funções matemáticas
- numpy, para processamento numérico
- scipy, para cálculos científicos

Algo sobre “variáveis“ em Python

Um aspecto importante de Python pode ser esclarecido pelo uso do type()

Algo sobre “variáveis“ em Python

Dentro do modo iterativo tecle

a = 1

b = 1.

c = “abc“

type(a)

type(b)

type(c)

Algo sobre “variáveis“ em Python

Teremos o que se segue

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "copyright", "credits" or "license()" for more information.
>>> a = 1
>>> b = 1.0
>>> c = "abc"
>>> d = True
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'bool'>
>>>
```

Algo sobre “variáveis“ em Python

Observe que temos uma descrição do tipo do argumento e descobrimos que o que chamamos até agora de tipo de variáveis tem em Python um outro nome: **classe**

Algo sobre “variáveis“ em Python

Este é uma das facetas do Python, ou melhor, um dos paradígmas de programação que ele inclui: Orientação a Objetos

No entanto, sigamos em frente e não se preocupe com isto no momento