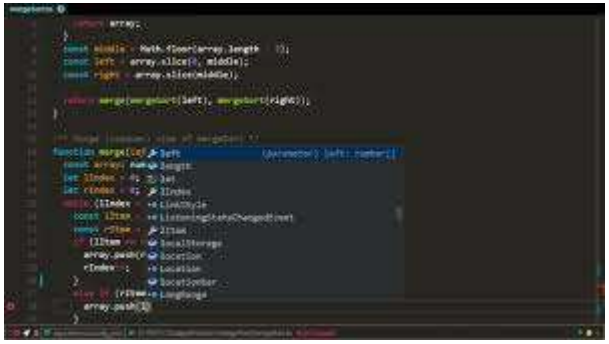
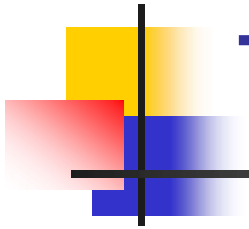
SME0827 - Estruturas de Dados



Python 2

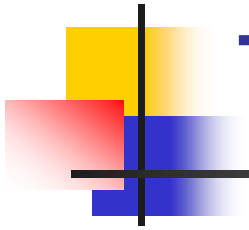
Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE:
Monitor:



Temas deste módulo

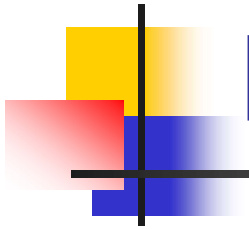
- Variáveis
- Expressões
- Comandos
- Funções

Variáveis



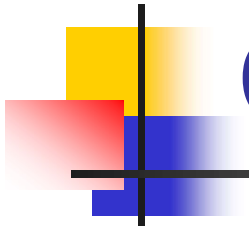
Tópicos

- Dados
- Constantes
- Variáveis
- Tipos de dados
- Números
- *Strings*



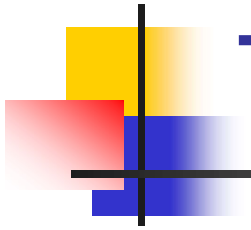
Dados

- Uma das principais características dos programas é que eles manipulam dados
  - Programas precisam armazenar dados
    - Valores dos dados não mudam: constantes literais
    - Valores podem mudar: variáveis
      - Cada variável armazena valores de um tipo de dado
  - Existem vários tipos de dados diferentes
    - Números
    - Textos
    - Tipos mais complexos



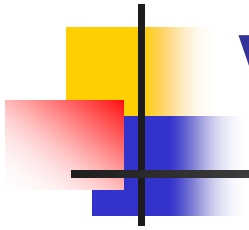
Constantes literais

- Valor é usado literalmente (não são variáveis)
- Facilita uso e manutenção de um valor
- Exemplos
  - *7*
  - *"texto constante"*
  - *4.765*
- Ao contrário de outras linguagens, Python não tem declaração de constantes
  - Na linguagem C, *#define minimo 0*



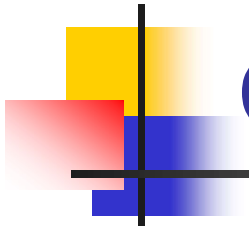
Tipagem dinâmica

- Usada por Python
- Valor da variável define seu tipo
  - Tipo da variável pode mudar durante execução de um programa
    - Uma variável pode armazenar um valor inteiro em um ponto de um programa
      - E armazenar um *string* mais adiante
      - E um valor real depois
      - ...



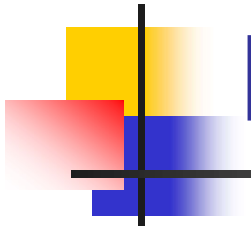
Variáveis

- Sintaxe para declarar variáveis em Python:
  - *Nome-da-variavel = valor*
- Podem ser declaradas em qualquer local do código
- Propriedades de variáveis:
  - Nome
  - Tipo
  - Tempo de vida
  - Escopo



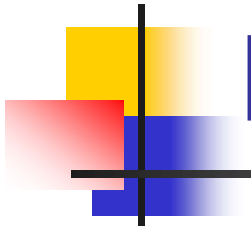
Convenções para nomes de variáveis

- Deve iniciar com letra ou *underscore* (_)
  - Demais caracteres de um nome devem ser letras, números ou *underscores*
  - MAIÚSCULAS ≠ minúsculas
  - Nomes longos dificultam leitura de expressões
- Bons nomes reduzem necessidade de comentários
- Não utilizar palavras-chave para nomes
- Também valem para funções



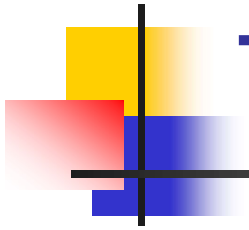
Palavras-chave (*keywords*)

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



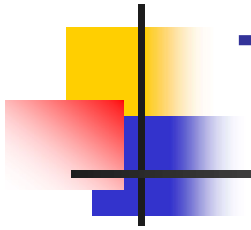
Palavras-chave (*keywords*)

- Python tem uma biblioteca para elas
- Não precisa memorizar quais são
  - *keyword.iskeyword(aaa)*
    - Retorna se *aaa* é uma palavra chave de Python
  - *keyword.kwlist*
    - Retorna todas as palavras chave de Python



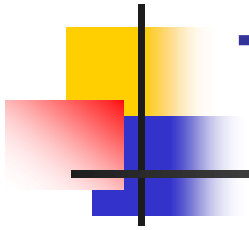
Tipos de dados

- Definidos pelo programador
  - Classes (orientação a objetos)
- Tudo em Python é um objeto
  - Incluindo números, *strings* e funções



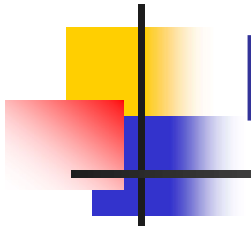
Tipos numéricos

- Semelhantes aos tipos numéricos de outras linguagens
  - Inteiro (*int*)
  - Ponto flutuante (*float*)
  - Número complexo (*complex*)



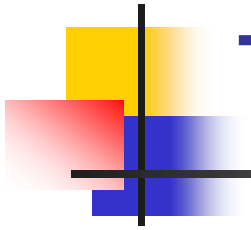
Tipo inteiro

- Inteiro normal (*int*)
  - Precisão ilimitada
    - Pode utilizar tantos bits quanto necessário
      - Ao contrário da maioria das linguagens
  - Tipo booleano (*bool*)
    - Subtipo de inteiro
    - Adicionado a Python na versão 2.3
      - Versões anteriores não usavam um tipo booleano específico, mas os valores 1 e 0



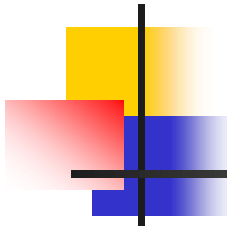
Exemplo

```
>>> 2+2
4
>>> # Isso é um comentário
>>> 2+2 # Agora um comentário na mesma linha
4
>>> (50-5*6)/4
5
>>> # Divisão de inteiro por inteiro:
>>> 7/3
2.3333333333333335
>>> 7/-3
-2.3333333333333335
```



Tipo Booleano

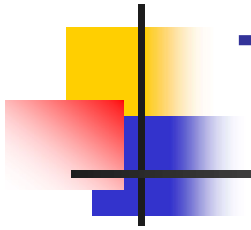
- Contém apenas dois valores:
  - *True* (verdadeiro)
  - *False* (falso)
- Equivale aos valores 0 (*False*) e 1 (*True*)
  - X == 0 e X != 0
- Podem ser combinados com valores numéricos em expressões aritméticas
- Conversão para *strings* retorna os valores "*True*" ou "*False*"



Exemplos

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool([])
False
>>> bool (-21)
True
>>> bool( (1,) )
True
```

```
>>> True + 1
2
>>> False + 1
1
>>> False * 9
0
>>> (True+True) * 7
14
>>> x = 3
>>> 6 + (x!=0)
7
```

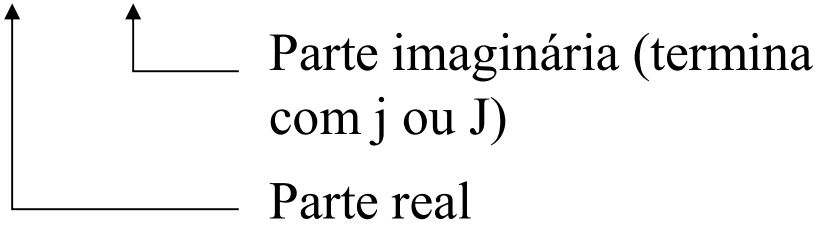


Tipo ponto flutuante

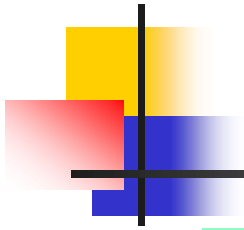
- Números reais em ponto flutuante (*float*)
- Aumenta precisão em relação a *int*
  - Às custas de mais espaço de memória
- Mesmas regras para escrever valores de ponto flutuante (*double*) da linguagem C
  - Ex. 4.01, 2.34e+9, 4E120
    - Notação E indica potência de 10
      - 4.7E-3 = $4.7 * 10^{-3}$

Tipo números complexos

- Escritos na forma *5 + 7j* ou *complex (5, 7)*



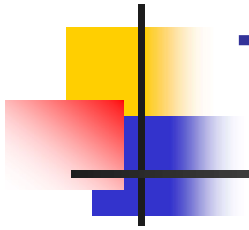
- Parte real é opcional
  - Parte imaginária pode vir no início
- Implementado como um par de números do tipo *float*
  - Permite uso de operações numéricas para números complexos



Exemplo

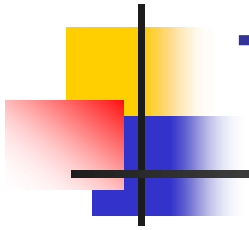
```
>>> 1j * 1j  # imaginários
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

```
>>> (3+4j).real
3.0
>>> (3+4j).imag
4.0
>>> a =1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```



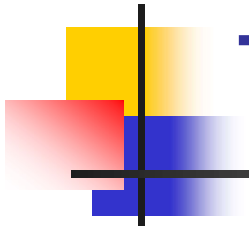
Tipo *string*

- Coleção ordenada de caracteres
  - Ordem pré-definida de caracteres, da esquerda para a direita
  - Ex.: “Ciencia”
- Usado para armazenar e representar sequências de caracteres
- Subtipo do tipo *sequence*
  - Inclui também listas e tuplas



Tipo *string*

- Pode representar
  - Nomes
  - Frases
  - Textos
  - Sequências de DNA
  - Códigos
- Python possui vários operadores para manipular *strings*

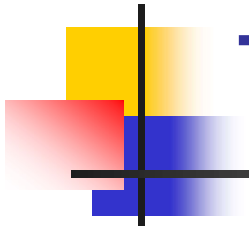


Tipo *string*

- Um *string* pode ser encaixado entre aspas simples ou duplas
  - Ex.: *"Tudo bem?" = ´Tudo bem?´*
  - Usa-se aspas triplas, ''' ou """, para *strings* de várias linhas
    - Aspas duplas ou simples podem ser usadas dentro de um *string*

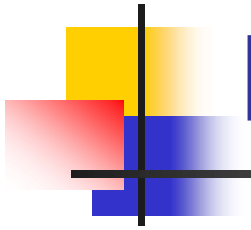
```
'''This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
'''
```

Expressões



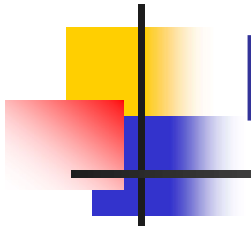
Tópicos

- Expressões
- Operadores
  - Aritméticos
  - Relacionais
  - *Bitwise*
  - Atribuição
- Mistura de tipos



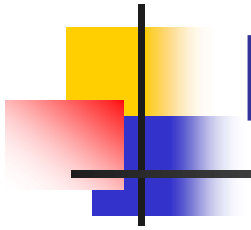
Expressões

- Construídas de acordo com a sintaxe da linguagem
- Calculam valores que podem:
  - Ser atribuídos a variáveis
  - Controlar fluxo de execução de um programa
- Valor e tipo do resultado
  - Dependem do operador e do tipo dos operandos utilizados



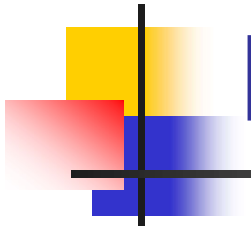
Exemplos

```
>>> 3 + 4
7
>>> 3 * 4
12
>>> x = 4 * 3
>>> x
12
>>> x = - 7 + 5 * 3 / 4 - (8 * 3 + 2)
>>> x
```



Exemplos

```
>>> 3 + 4
7
>>> 3 * 4
12
>>> x = 4 * 3
>>> x
12
>>> x = - 7 + 5 * 3 / 4 - (8 * 3 + 2)
>>> x
-29.25
```



Programa em Python

```
# Cálculo do valor de uma expressão

val = 0
while (val < 16):
    val = val * 3 - 8 + 14
print ('Valor da expressão eh: %d\n' %(val))
```

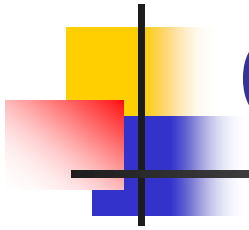
Mesma programa em C

```
/* Calculo do valor de uma expressao*/
# include <stdio.h>

main (){
    int valor;

    valor = 0;
    while (valor < 16){
        valor = valor + 4;
    }
    printf ("Valor da expressao eh: %d\n", valor);
}
```

Operadores aritméticos

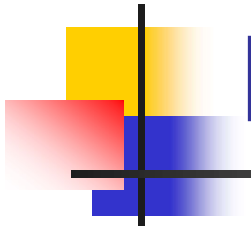
- Operadores binários

Operador	Uso	Descrição
+	*op1 + op2*	*Adiciona op1 e op2*
-	*op1 - op2*	*Subtrai op2 de op1*
*	*op1 * op2*	*Multiplica op1 por op2*
/	*op1 / op2*	*Divide op1 por op2*
//	*op1 // op2*	*Divide op1 por op2 e trunca*
%	*op1 % op2*	*Calcula o resto de op1 / op2*
**	*op1 ** op2*	*Calcula op1 elevado a op2*



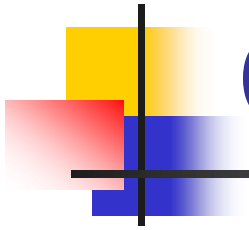
Operadores de divisão

- Python tem dois operadores de divisão
  - Divisão /
    - Resultado é um valor real
      - Não importa se a divisão for exata
      - Exemplos:
        - 9/4 = 2.5
        - 8/4 = 2.0
  - Divisão e truncamento //



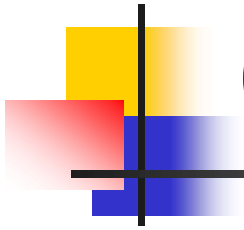
Divisão e truncamento (//)

- Parte fracionária é descartada (truncamento)
  - Se os dois operandos forem do tipo inteiro
    - Resultado é do tipo inteiro
    - Ex.: 9 // 4 = 2
  - Se pelo menos um dos operandos for do tipo ponto flutuante
    - Os argumentos são primeiro convertidos para o tipo de maior precisão (neste caso, *float*)
    - Resultado é do tipo ponto flutuante
    - Ex. 9.0 // 4 = 9 // 4.0 = 9.0 // 4.0 = 2.0



Operador de resto (%)

- Resto da divisão de um valor por outro
  - Exemplos:
    - 9 % 4 =
    - 9.0 % 4 =
    - 7 % -4 =
    - -7 % 4 =
  - Útil para testar se um número é divisível por um outro (resto da divisão é igual a zero)
  - Resultado possui mesmo sinal do segundo operando
    - Valor absoluto do resultado é menor que o valor absoluto do segundo operando



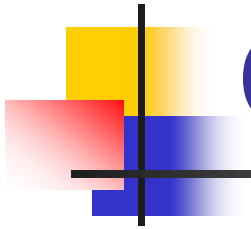
Operador de resto (%)

- Resto da divisão de um valor por outro
  - Exemplos:
    - 9 % 4 = 1
    - 9.0 % 4 = 1.0
    - 7 % -4 = -1
    - -7 % 4 = 1
  - Útil para testar se um número é divisível por um outro (resto da divisão é igual a zero)
  - Resultado possui mesmo sinal do segundo operando
    - Valor absoluto do resultado é menor que o valor absoluto do segundo operando

Operadores relacionais

- Comparam dois valores e retornam a relação entre eles
  - Retornam valor do tipo Booleano

Operador	Uso	Descrição
==	*op1 == op2*	*op1 é igual a op2*
!=	*op1 != op2*	*op1 é diferente de op2*
>	*op1 > op2*	*op1 é maior que op2*
<	*op1 < op2*	*op1 é menor que op2*
>=	*op1 >= op2*	*op1 é maior ou igual a op2*
<=	*op1 <= op2*	*op1 é menor ou igual a op2*



Operadores relacionais

- Retornam um dentre dois valores
  - *True* ou *False*
    - Qualquer valor ≠ 0 é interpretado como *True*
      - Valor diferente de 1 e 0 pode ser útil, mas deve ser evitado
  - Valores especiais do tipo *bool* (não são *strings*)
- Não confundir = (atribuição) com == (igual)
  - Um dos erros mais comuns
  - Interpretador Python geralmente não detecta esses erros

Operadores lógicos

- Utilizam operandos Booleanos
  - Retornam resultado Booleano

Precedência

Operador	**Uso**	Retorna **True** se
not	not op1	op1 é falso
and	op1 and op2	op1 e op2 são ambos verdade, avalia condicionalmente op2
or	op1 or op2	op1 ou op2 é verdade, avalia condicionalmente op2

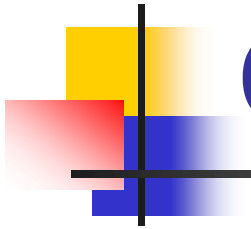
Maior

Menor

Operadores lógicos

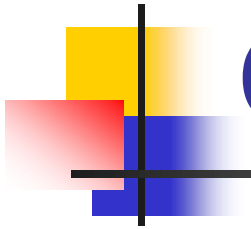
- Como funcionam

Operandos		Operadores		
op1	op2	not (op2)	and	or
False	*False*	*True*	*False*	*False*
False	*True*	*False*	*False*	*True*
True	*False*	*True*	*False*	*True*
True	*True*	*False*	*True*	*True*



Operadores lógicos

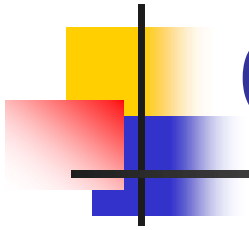
- Avaliação preguiçosa
  - Operandos de um operador lógico são avaliados da esquerda para a direita
  - Avaliação termina assim que a resposta puder ser determinada
    - op1 *and* op2
    - op1 *or* op2



Operadores bitwise

- Permite manipular bits dos dados

Operador	Uso	Descrição
>>	*op1 >> op2*	*move op2 posições p/ direita os bits de op1*
<<	*op1 << op2*	*move op2 posições p/ esquerda os bits de op1*
&	*op1 & op2*	*bitwise and*
|	*op1 | op2*	*bitwise or*
^	*op1 ^ op2*	*bitwise xor (ou exclusivo)*
~	*~ op2*	*complemento a 1*



Operadores bitwise

- Como funcionam bit a bit

Operandos		Operadores			
op1	op2	~op2	&	|	^
0	*0*	*1*	*0*	*0*	*0*
0	*1*	*0*	*0*	*1*	*1*
1	*0*	*1*	*0*	*1*	*1*
1	*1*	*0*	*1*	*1*	*0*

Operadores bitwise

- Como funcionam para um byte

Operador	Exemplo	Resultado
>>	*11110011 >> 2*	
<<	*11110011 << 1*	
&	*10001001 & 00000101*	
|	*10001001 | 00000101*	
^	*10001001 ^ 00000101*	
~	*~11110000*	

Operadores bitwise

- Como funcionam para um byte

Operador	Exemplo	Resultado
>>	*11110011 >> 2*	*00111100*
<<	*11110011 << 1*	*11100110*
&	*10001001 & 00000101*	*00000001*
|	*10001001 | 00000101*	*10001101*
^	*10001001 ^ 00000101*	*10001100*
~	*~11110000*	*00001111*

Operador (comando) de atribuição

- Atribuição de valores a variáveis ocorre por meio de uma expressão
  - Tipo da variável será o tipo do operando atribuído a ela
  - Ex.: resultado = 4
    - Variável será do tipo *int*

```
>>> n = 3
>>> n
3
>>> n = 2.0
>>> n
2.0
```

Operador de atribuição

- Permite atribuir valores a mais de uma variável simultaneamente

```
>>> a, b, c = 3, 4, 5
>>> a

>>> b

>>> c
```

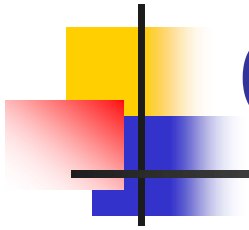
```
>>> x = y = z = 0
>>> x

>>> y

>>> z
```

Operador de atribuição

- Permite atribuir valores a mais de uma variável simultaneamente

```
>>> a, b, c = 3, 4, 5
>>> a
3
>>> b
4
>>> c
5
```

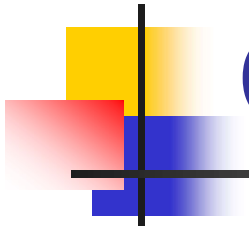
```
>>> x = y = z = 0
>>> x
0
>>> y
0
>>> z
0
```



Operador de atribuição

- Python permite a combinação da atribuição com operadores binários
  - Atribuição com atalho

    total += 3 $\Leftrightarrow$ *total = total + 3*
  - Exemplos:
    - *valor+= 3*
    - *res -= 2*
    - *x /= 10*
    - *y //= 8*
    - *z *= 2*



Conclusão

- Variáveis
- Expressões

Perguntas