Revisão de Python

Prof. Saulo Oliveira
Técnico em Informática para Internet
Instituto Federal do Ceará

Introdução

O algoritmo é uma sequência de passos lógicos e finitos que permite solucionar problemas;

- O objetivo de aprender a criar algoritmos é que este é a base de conhecimentos para as linguagens de programação;
- Em geral, existem muitas maneiras de resolver o mesmo problema. Ou seja, podem ser criados vários algoritmos diferentes para resolver o mesmo problema;
- Assim, ao criarmos um algoritmo, indicamos uma dentre várias possíveis sequências de passos para solucionar o problema.

Propriedades essenciais

Completo	Sem redundância	Determinístico	Finito
Todas as ações precisam ser descritas e devem ser únicas.	Um conjunto de instruções só pode ter uma única forma de ser interpretada.	Se as instruçõess forem executadas, o resultado esperado será sempre atingido.	As instruções precisam terminar após um número limitado de passos.

Instruções e Tipos de dados

As informações manipuladas pelo computador podem ser classificadas em:

Instruções	Dados
Coordenam o funcionamento do computador, determinando a maneira como os dados devem ser tratados.	São as informações a serem processadas pelo computador.

Em Python, os dados podem ser dos tipos: numéricos (`int` e `float`), lógicos (`bool`), literais (`string`), listas (`list`) e dicionários (`dict`). Há outros, mas deixemos para depois!

Exercício

Classifique os dados abaixo de acordo com seu tipo, assinalando com `I` os dados do tipo inteiro, com `R` os reais, com `L` os literais, com `B` os lógicos (booleanos), e com `N` aqueles para os quais não é possível definir a priori um tipo de dado.

	0.21		"josé"		"+3257"		True
	1		0,35		+3257.		False.
	V		TRUE		"-0.0"		"abc"
	"0"		+3257		"False"		+36
	1%		'a'		True		± 3

Declaração de variáveis

Em Python é necessário apenas o nome da variável, seguido do símbolo = e o valor que ela irá armazenar. O tipo da variável será o mesmo tipo de dado que ela armazena.

```
idade = 32
preco = 100.21
teste = True
nome = "SAULO"
```

Regras de nomenclatura (o nome das variáveis):

- Podem ter dígitos, letras maiúsculas e minúsculas, e underscore (_);
- Não pode ser iniciado por dígito e não são permitidos espaços em branco e nem podem conter caracteres especiais (@, $, +, -, %, !, /, ?, #);
- Nem palavras reservadas (`keywords`).

Saída de dados

> Usaremos a função `print`.

- A função `print` mostra em formato texto para o usuário o conteúdo de um variável;
- Também pode mostrar strings ou combinações de strings e variáveis, bastando separar por (,) vírgula.
- Podemos usar o print com valores dos tipos primitivos;
- Podemos executar expressões e só o valor do resultado vai ser impresso.

```
idade = 32
nome = 'Saulo'
print('Meu primeiro programa')
print(f'Meu nome é {nome} e minha idade é:', idade)
print('Faltam', 65 - idade, 'anos para eu me aposentar!')
```

Entrada de dados

> Usaremos a função `input`.

- A função `input` requer um texto que será mostrado para o usuário e retorna o que o usuário digitou (sempre do tipo literal `str`, precisamos converter depois, se necessário).
- Guardamos o valor retornado pela função `input` em uma variável.

```
nome = input('Digite seu nome: ')
print(f'Seja muito bem-vindo(a), {nome}!')

idade = int(input('Digite sua idade: '))
print(f'Legal, que você tem {idade} anos!')

altura = float(input('Digite sua altura: '))
print(f'E com altura de {altura} metros.')
```

Entrada de dados descasada

```
idade = input('Me diga novamente quantos anos tem:')
proxima_idade = idade + 1
print(f'Ano que vem, você terá {proxima_idade} anos')
```

QUAL O PROBLEMA COM O CÓDIGO ACIMA?

Conversão de tipos

A conversão de tipos é o ato de forçar uma expressão a utilizar e retornar um determinado tipo. Podemos ter dois tipos de conversões de tipos, pode ser implícita ou explicitas, que são conversões especificadas.

> Para saber o tipo de dado de uma expressão ou variável, basta usarmos a função `type`.

Exercício

Declaração	Tipo da variável	Conversão	Resultado	Tipo do resultado
`x = "42"`		`int(x)`		
`n = 123`		`str(n)`		
`pi = 3.14`		`int(pi)`		
`saldo = 567`		`float(saldo)`		
`tem_pix = True`		`str(tem_pix)`		
`poupanca = 0.0`		`bool(poupanca)`		
`nome = 'Saulo'`		`bool(nome)`		
`cpf = ''`		`bool(cpf)`		
`faltei = False`		`int(faltei)`		
`merenda = True`		`float(merenda)`		

Exercício prático

1. Faça um algoritmo para converter uma temperatura dada em Fahrenheit para Celsius.
2. Faça um algoritmo que receba duas notas e seus respectivos pesos, calcule e mostre a média ponderada.
3. Faça um algoritmo que receba um valor referente a uma compra em dólar no cartão de crédito, calcule e mostre o valor de conversão para real. Além disso, sabendo que em compras internacionais incide-se O IOF sobre o total, adicione o valor da taxa (valor de 6,38%). Ademais, adote o valor do dólar R$ 5,00.

Operadores e Expressões - I

Operadores aritméticos.

> São operadores matemáticos básicos que envolvem números e são utilizados para realizar cálculos e manipular quantidades numéricas.

Operador	Uso	Comentário
`+`	`x + y`	Soma o conteúdo de `x` e de `y`.
`-`	`x - y`	Subtrai o conteúdo de `y` do conteúdo de `x`.
`*`	`x * y`	Multiplica o conteúdo de `x` por pelo conteúdo de `y`.
`/`	`x / y`	Divide o conteúdo de `x` pelo conteúdo de `y`.
`%`	`x % y`	Obtém o resto da divisão de `x` por `y`.
`//`	`x // y`	Obtém o quociente inteiro da divisão de `x` por `y`.
`**`	`x ** y`	Eleva o conteúdo de `x` à potência do conteúdo de `y`.

Operadores e Expressões - II

Operadores relacionais.

> São operadores que comparam valores para determinar relações como igualdade, desigualdade, maior ou menor, retornando um valor lógico (`True` ou `False`).

Operador	Uso	Comentário
`==`	`x == y`	O conteúdo de `x` é igual ao conteúdo de `y`.
`!=`	`x != y`	O conteúdo de `x` é diferente do conteúdo de `y`.
`<=`	`x <= y`	O conteúdo de `x` é menor ou igual ao conteúdo de `y`.
`>=`	`x >= y`	O conteúdo de `x` é maior ou igual ao conteúdo de `y`.
`<`	`x < y`	O conteúdo de `x` é menor que o conteúdo de `y`.
`>`	`x > y`	O conteúdo de `x` é maior que o conteúdo de `y`.

Operadores e Expressões - III

Operadores Lógicos.

> Operadores lógicos permitem a realização de operações lógicas sobre valores booleanos. Esses operadores geralmente são utilizados para tomar decisões condicionais e controlar o fluxo de execução de um programa.

Operador	Uso	Comentário
`not`	`not x`	Equivale a modificar o conteúdo de `x` pela negação.
`and`	`x and y`	Retorna `True` se e somente se `x` e `y` forem `True`. Caso contrário, retorna `False`.
`or`	`x or y`	Retorna `False` se e somente se `x` e `y` forem `False`. Caso contrário, retorna `True`.

Precedência

A precedência determina a ordem em que os operadores são avaliados em uma expressão, quando ela envolve múltiplos operadores.

É possível também alterar a ordem de avaliação usando parênteses.

Operadores	Descrição
**	Exponenciação.
+, -	Operadores unários (modificam o sinal).
*, /, %, //	Produto, divisão, resto da divisão e divisão inteira.
+, -	Adição e subtração.
<=, <, >, >=	Operadores de comparação.
==, !=	Operadores de igualdade.
not, and, or	Operadores lógicos.

Exercícios

Dada a seguinte expressão:

```
2 - 2 ** 3 * 2
```

1. Adicione um conjunto de parênteses para que a expressão seja avaliada como -12.
2. Agora mova seus parênteses para que a expressão seja avaliada como -62.
3. Mova seus parênteses uma última vez para que a expressão seja avaliada como 0.

Exercícios

Dada a seguinte expressão:

```
2 / 3 * 4 ** 2
```

- Adicione dois conjuntos de parênteses que deixam o valor da expressão inalterado. Os parênteses não podem incluir a expressão inteira nem um único número.

QUANTO VALE A EXPRESSÃO ABAIXO?

```
-2 ** 2
```

Exercícios

Analise o programa abaixo e, para cada uma das saídas (comandos `print`), detalhe passo a passo como o Python (segundo suas prioridades) resolveria as equações e o resultado final obtido.

```
x = 2
y = 3
z = 0.5

print(x + x * x ** (y * x) / z)

print(not x + z < y or x + x * z >= y and True)

print(x + y == z)
```

Exercícios

Indique o resultado das expressões mostrando passo a passo a ordem de avaliação, sendo: `x = 6.0`, `y = 2`, `z = 4.0`, `a = 8`, `b = 7.5`, `c = 12`. Indique quando ela não puder ser realizada e informe por qual motivo.

a) `x - y * (a + 1) == z * -c`

b) `x - y * a > c % y`

c) `c % y <= y % c`

d) `(b * 4) >= (a + a * 2) and a >= c ** - 2`

e) `a + 3 > -b + -c`

f) `b + a > c + c and a != c < b != a`

g) `a // c < (b % 2) or (c ** b * 3) < a * 3`

Condicionais

- Condicionais em lógica de programação são estruturas que permitem a execução de diferentes blocos de código dependendo da avaliação de uma condição específica.
- Essas estruturas são fundamentais para controlar o fluxo de execução de um programa, permitindo que partes específicas do código sejam executadas ou ignoradas com base em condições lógicas;
- Eles avaliam uma expressão booleana e, com base no resultado (verdadeiro ou falso), direcionam o programa para diferentes caminhos de execução.

Condicional simples

> Permite a execução condicional de um bloco de código.

Avalia-se uma expressão booleana e, se a condição for verdadeira (`True`), o bloco de código indentado após o `if` é executado; caso contrário, o bloco é ignorado.

```
idade = int(input('Digite sua idade: '))

if idade > 18:
    print('Você é maior de idade')

saldo = float(input('Digite seu saldo: '))

if saldo > 0:
    print('Você não está liso!')
```

Exercícios

1. Faça um programa que receba dois números e retorne o valor do maior.
2. Faça um programa que solicite ao usuário dois números e imprima "São iguais" se forem iguais; caso contrário, imprima "São diferentes".
3. Crie um programa que verifica se um número é múltiplo de 3 e de 5 e imprime mensagens correspondentes.
4. Peça ao usuário para digitar uma palavra. Se a palavra é "python", imprima uma mensagem de confirmação ou correção (você errou a senha secreta).
5. Crie um programa em Python que solicite um número ao usuário. Verificar se o número é um múltiplo de 10 e imprima uma mensagem apropriada.
6. Faça um programa que recebe a média final de um aluno e mostra sua situação, que pode ser aprovado (M ≤ 7), reprovado (M < 4) e AF (4 ≤ M < 7).

Condicional composto

> Já na estrutura composta, existe um bloco de código que executa para cada um dos possíveis resultados (`True` ou `False`).

Avalia-se uma expressão booleana e, se a condição for verdadeira (`True`), o bloco de código indentado após o `if` é executado; caso contrário, o bloco após o `else` é executado.

```
ano = int(input('Digite o ano: '))

if ano % 2 == 0:
    print('O ano é par!')
else:
    print('O ano é ímpar!')
```

Exercícios

1. Faça um programa para receber dois números positivos e mostre-os em ordem crescente.
2. Faça um programa para ler dois números inteiros e informar se estes números são iguais ou diferentes.
3. Faça um programa que receba um número e imprima se ele é positivo ou negativo.
4. Solicite ao usuário sua renda mensal e o valor desejado para empréstimo. Imprima se o empréstimo pode ser ralizado, sabendo que somente é aprovado caso o valor desejado seja menor do 5x o valor da renda mensal.

Condicional encadeado

> Já na estrutura composta encadeada, somente uma condição pode ocorrer dentro do encadeamento.

Permite avaliar condições adicionais se as anteriores forem falsas. A instrução `elif` é a junção das instruções `else` e `if`. Leia como se fosse *senão se*.

```
nota = float(input('Digite sua nota na disciplina'))

if nota >= 7:
    print('Aprovado.')
elif nota < 4:
    print('Reprovado.')
else:
    print('Prova final.')
```

Exercícios

Faça um algoritmo que mostre a classificação do IMC de um indivíduo. O peso é dado em quilogramas (Kg) e a altura em metros (m), conforme a fórmula que segue:

$$\text{IMC} = \frac{peso}{altura^2}$$

IMC	Classificação do IMC
< 16	Magreza grave
16 a < 17	Magreza moderada
17 a < 18,5	Magreza leve
18,5 a < 25	Saudável
25 a < 30	Sobrepeso
30 a < 35	Obesidade Grau I
35 a < 40	Obesidade Grau II (severa)
> 40	Obesidade Grau III (mórbida)

Estruturas de repetição

> Essas estruturas são fundamentais para automatizar tarefas repetitivas e controlar o fluxo de execução de um programa.

As repetições **podem ser de uma quantidade fixa** ou **depender de uma condição**.

Comando `for` é utilizado quando o número exato de iterações é conhecido antecipadamente. Geralmente, emprega-se uma variável de controle que percorre uma sequência ou intervalo predefinido.

Já o comando `while` é utilizado quando o número de iterações não é conhecido antecipadamente e depende de uma condição. Assim, o bloco de código é repetido enquanto a condição especificada for verdadeira.

Computador chinês

- Elaborar uma tabela onde cada linha se refere a cada variável envolvida e o resultado de uma operação em particular, ou número da linha do algoritmo (ou observação pertinente);
- Executar os passos previstos no algoritmo;
- Verificar se os resultados obtidos são coerentes. Senão, corrigir o algoritmo e testar novamente para as entradas anteriores;
- Realizar o teste para diferentes entradas e concluir quando todos os testes forem bem sucedidos.

Exemplo do Computador chinês

```
x = 1

while  x < 10:

  print( x )

  x = x + 2
```

Variável	Valor	1	2	3	4	5
x						

Exemplo do Computador chinês

```
x, y = 1, 5

while  x < y:

  y = y * 2
  x = x + y

  print(x, y)
```

Variável	Valor	1	2	3	4	5
x	1					
y	5					

Laços infinitos

> É preciso ter muito cuidado com estruturas de repetição.

Sempre que você for escrever uma estrutura de repetição, certifique-se de que a condição será avaliada como falsa em algum momento da execução do laço. Se isso não acontecer, há o risco de o *loop* ser executado indefinidamente (*loop* infinito).

```
x = 1

while  x < x + 1:
    x = x + 1
    print( x )

print('Acabou!??')
```

Exercícios

1. Faça um algoritmo para verificar se o usuário é maior de idade pedindo somente o ano de nascimento. O programa só deve encerrar quando ele digitar um ano que corresponda a maior idade.
2. Escreva um algoritmo de troca de senhas. Ele deve pedir a senha antiga e a senha nova. O algoritmo só encerra quando as senhas forem diferentes.
3. Escreva um algoritmo de troca de senhas mais robusto. Ele deve pedir a senha antiga, pedir a senha nova e a confirmação a senha nova. O algoritmo só deve encerrar sob a condição da senha antiga ser diferente da nova e a confirmação da senha for igual a senha nova.

Strings

> Representadas por dados do tipo texto e podem ser utilizadas para armazenar e manipular informações, como palavras, frases ou dados textuais.

- Uma `str` se comporta como uma sequência;
- Em Python, uma sequência é um tipo de dado composto, que armazena um conjunto de elementos em uma ordem específica;
- No caso de uma `str`, os elementos são os caracteres;
- Cada caractere entre as aspas ocupa uma **posição** (**índice**) e pode ser acessado usando esse **índice** e colchetes.

Acesso por índice

> Os índices sempre iniciam por 0.

Índice neg.	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Caractere	S	a	u	l	o		O	l	i	v	e	i	r	a

```
nome = 'Saulo Oliveira'
print(nome[4])
print(nome[3])
print(nome[13])
```

```
nome = 'Saulo Oliveira'
print(nome[-8])
print(nome[-3])
print(nome[-130])
```

Fatiamento

> Se quisermos usar apenas parte de uma string, podemos usar fatiamento;

O fatiamento funciona utilizando dois pontos `:` na notação de colchetes para indicar um intervalo.Sintaxe: `VARIAVEL[ INICIO:FIM ]`.

> ⚠️ O caractere de início entra, mas o de fim não!

```
nome_completo = 'Saulo Oliveira'
nome = nome_completo[0:5]
sobrenome = nome_completo[6:14]

print(nome)
print(sobrenome)

nome = nome_completo[:5]
sobrenome = nome_completo[6:]
```

```
s = 'HelloWorld'
reverse_str = s[::-1]
print(reverse_str)

s1 = s[2:8:2]
print(s1)

s2 = s[8:1:-1]
print(s2)
```

Operadores de ocorrências

> O operador `in` permite verificar se uma string **está contida** em outra. Já o operador `not in` permite verificar se uma string **não está contida** em outra.

```
ditado = 'Quem tem boca vai a roma'

print('boca' in ditado)
print('a' in ditado)
print('quem' in ditado)

print('a' not in ditado)
print('quem tem' not in ditado)
```

Imutabilidade em `str`

> Strings são imutáveis! Uma vez criada uma string, você não pode alterá-la (**mudar seu conteúdo**).

```
>>> nome = 'Saulo Oliveira'
>>> nome[0] = 'P'
>>> print(nome)
```

```
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
TypeError: 'str' object does not support item assignment
```

Concatenação

> Juntar duas ou mais strings (ou partes de strings) em uma nova string. Usaremos o operador `+`.

```
nome = 'Saulo'
sobrenome = 'Oliveira'
nome_completo = primeiro + ultimo

print(nome_completo)
```

Um caso especial de concatenação é a repetição de uma string várias vezes.

```
palavra = 'blá'
repetida = palavra * 5

print(repetida)
```

Interpolação

> Interpolação de string é o **processo de avaliação de uma string literal** contendo um ou mais **espaços reservados** (*placeholders*), produzindoum resultado no qual os **espaços reservados são substituídos por seus valores correspondentes**.

Tais strings serão chamadas de "f-strings" ou "strings formatadas".

```
nome = 'Saulo'
reais =  5

print(f'O professor {nome} é o melhor!')
print(f'Só tenho {reais} no pix.')

msg = f'O {nome} tem de saldo R$ {reais}'
print(msg)
```

Exercícios

1. Crie uma nova string utilizando concatenação para adicionar a palavra "dia", resultando em "Bom dia".
2. Utilize concatenação para criar uma string que represente a soma dos números da seguinte forma: "A soma de 10 e 5 é 15."

```
numero1 = 10
numero2 = 5
```

3. Suponha as variáveis abaixo. Utilize a interpolação de variáveis para criar uma string que exiba: "O preço do Notebook é R$ 2500.99."

```
produto = "Notebook"
preco = 2500.99
```

Alguns métodos de `str`: Caixa do texto

> Para executá-los, você não precisa utlizat a sintaxe `str.método(*args)`.

- `str.lower(): -> str`: retorna uma cópia da string com tudo em minúsculo;
- `str.upper(): -> str`: retorna uma cópia da string com tudo em maiúsculo.

```
nome = 'Saulo Oliveira'
M = nome.upper()
m = nome.lower()
print(M)
print(m)
print(f'o {nome} é o melhor prof!'.upper())
print('IFCE'.lower())
```

Verificação de prefixo e de sufixo

- `str.startswith(str): -> bool` : verifica se uma string começa com o prefixo dado.
- `str.endswith(str): -> bool` : verifica se uma string termina com sufixo dado.

```
nome = 'Saulo Oliveira'

print(nome.startswith('saulo'))
print(nome.startswith('Saulo'))
print(nome.startswith('Oliveira'))

print(nome.endswith('Oliveira'))
print(nome.endswith('Oliveir'))
```

Quebra e cola de pedaços

- `str.split(str): -> list` : gera uma lista de strings com base na quebra de um delimitador.
- `str.join(list): -> str` : gera uma string a partir de pedaços, colando-os com o elemento concatenador.

```
>>> nome = 'Saulo Oliveira Professor IFCE Computação'
>>> print(nome.split())
['Saulo', 'Oliveira', 'Professor', 'IFCE', 'Computação']

>>> email = 'saulo.oliveira@ifce.edu.br'
>>> print(email.split('@'))
['saulo.oliveira', 'ifce.edu.br']

print('r'.join('aaa'))
print('r'.join(['a', 'a', 'a']))
```

Substituição

- `str.replace(str, str): -> str`: substitui todas as ocorrências de uma substring em uma string por outra substring. Ele retorna uma nova string com as substituições realizadas, sem modificar a string original.

```
>>> frase = "Python é uma linguagem de programação, e Python é divertido."
>>> nova_frase = frase.replace("Python", "Java")
>>> print(nova_frase)

Java é uma linguagem de programação, e Java é divertido.
```

Contagem e pesquisa

- `str.count(str): -> int` : conta o número de ocorrências **não-sobrepostas** de uma substring.
- `len(str): -> int` : Apesar de não ser um método e sim uma função `built-in` , conta o número de caracteres na string.

```
animal = 'arara'

print(animal.count('a'))
print(animal.count('r'))
print(animal.count('ara'))

print(len(''))
print(len('a'))
print(len('saulo'))
```

Exercícios

1. Faça um programa que receba um verbo regular no infinitivo e o conjugue no presente do indicativo. O programa deve aceitar todas as conjugações (1a, 2a e 3a).
2. Escreva um programa para obter uma string de uma determinada string (lida pelo usuário) em que todas as ocorrências de seu primeiro caractere são alteradas para '$', exceto o primeiro caractere em si.Exemplo: `restart`. Saída: `resta\$t`.
3. Partindo da variável abaixo, como você pode contar a quanitidade de letras total dessa lista.

```
>>> palavras = ["Python", "é", "uma", "linguagem", "poderosa"]
>>> ...
27
```

Listas

> Frequentemente, necessitaremos trabalhar com grandes coleções de dados e precisaremos guardar esses dados em memória, em variáveis.
>
> - Criar uma variável para cada item de dado é impraticável;
> - Utilizaremos variáveis compostas, chamadas vetores;
> - Um vetor é conhecido como variável **composta** (conjunto de dados com o mesmo nome), **homogênea** (variáveis do mesmo tipo), e **unidimensional** (acessadas por único índice);

⚠️ Em Python, usaremos o tipo `list` que implementam algumas dessas características -- exceto homogeneidade.

PEPs do Python

> Propostas de aprimoramento do Python.

Referências

https://en.wikipedia.org/wiki/String_interpolation

https://peps.python.org/pep-0498/