Minicurso de Python

Python

Nelson Carvalho Sandes

Centro de Ciências Tecnológicas - CCT
Universidade Federal do Cariri

2019

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Tópicos

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Tópicos

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Vantagens da linguagem Python

- Simples de compreender. Por conta disso, ela é uma boa opção para quem está iniciando o contato com programação.
- Apesar de simples, é bastante utilizada no mercado. Empresas como Google, Yahoo, NASA, Faceboook e Instagram usam Python em algumas aplicações.
- Também é referência na resolução de problemas científicos e matemáticos.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Tópicos

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Saída de dados

- Algumas vezes, algoritmos necessitam de dados do meio externo para processar e exibir a informação para o usuário.
- A recepção e exibição de dados para o usuário se dá a partir de dispositivos de entrada e saída (exemplo: teclado e monitor).
- Em Python, um meio comum de exibir dados para o usuário é através da instrução **print**.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Saída de dados

Primeiros códigos em Python

- Primeiro programa em Python:

```
print("Hello world!")
print("O rato roeu a roupa do rei de roma")
print("Algumas linguagens são mais simples do que outras")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Saída de dados

Primeiros códigos em Python

- Também é possível inserir informação no meio de um texto através da instrução **.format()**
- Por exemplo:

```
print("Ele tem 50 anos")
print("Ele tem {} anos".format(50))
```

```
print("A taxa SELIC é de 0.065")
print("A taxa {} é de {}".format('SELIC', 0.065))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Saída de dados

Operações matemáticas em Python

- Em Python, também é possível realizar e imprimir operações matemáticas.
- Por exemplo:

```
print(5 + 2)
print(3 - 2)
print(4 * 3)
print(8 / 2)
print(3 ** 2)
print(3 % 2)
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Saída de dados

Operações matemáticas em Python

- Caso necessário, podemos usar a instrução **format()** para deixar o programa mais informativo.

```
print("Soma: {}".format(5 + 2))
print("Subtração: {}".format(3 - 2))
print("Multiplicação: {}".format(4 * 6))
print("Divisão: {}".format(10 / 2))
print("Exponenciação: {}".format(2 ** 3))
print("Módulo: {}".format(3 % 2))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Saída de dados

Operações matemáticas em Python

- Em Python, divisão e multiplicaçao tem prioridade sobre soma e subtração:

```
print(2 + 6 / 2)
print(2 + 3 * 5 + 4)
print(4 / 2 + 3 * 2 + 1)
```

- Para dar prioridade para alguma operação, podemos fazer uso dos parênteses.

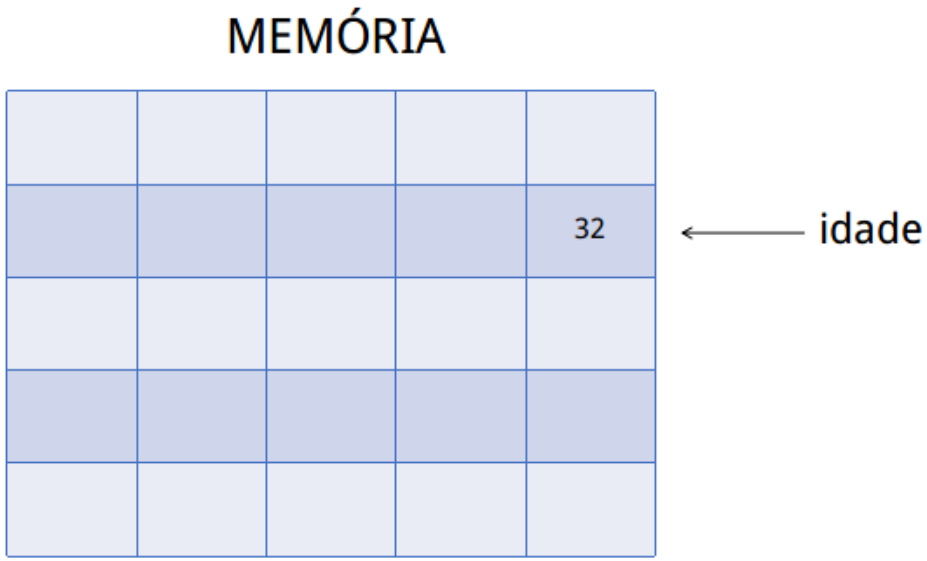
```
print((2 + 6) / 2)
print((2 + 3) * (5 + 4))
print(5 / (2 + 3) * (2 + 1))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Variáveis

Armazenamento em memória

- Em computação, o conceito de variável caracteriza-se por ser uma região da memória previamente identificada por um rótulo que possui a capacidade de armazenar certo valor, seja este numérico, alfabético, alfanumérico ou lógico.



UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Variáveis

Identificador

- O "nome" de uma variável é chamado de identificador.
- Regras para nomear uma variável:
  1. Iniciar com letra
  2. Não possuir caracteres especiais (#, %)
  3. Não possuir acentos (á, ú, â, ã)
  4. Não conter espaços (_ pode ser usado)
  5. Não pode ser uma palavra reservada (print, input, type)
- Apesar de não ser regra, é uma convenção não iniciar variáveis com letras maiúsculas. Outra convenção, é não usar todas as letras em maiúsculo. Como veremos posteriormente, esta é ma prática usada na definição de constantes em Python.

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Variáveis

Tipos de dados básicos

- Quando se diz "tipo de dado" o que se faz é determinar que certo valor armazenado em certa variável seja qualificado de acordo com sua característica de comportamento e estrutura.
- A linguagem Python suporta os seguintes tipos básicos de dados:
  1. **int** representa números inteiros.
  2. **float** representa números reais de ponto flutuante.
  3. **bool** representa valores lógicos (**True** e **False**).
  4. **str** representa um conjunto de caracteres.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Variáveis

Tipos de dados básicos

- Abaixo segue o exemplo da criação de uma variável de nome idade e do tipo inteiro.

```
idade = 32
print("O valor da variavel é: {}".format(idade))
```

- Outro exemplo:

```
idade = 32
altura = 1.69
nome = "Abraão"
print("{} tem {} anos e mede {}m".format(nome, idade, altura))
```

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Variáveis

Tipos de dados básicos

- Assim como números, também é possível fazer operações matemáticas com variáveis:

```
a = 4
b = 5
c = 2
print(a + b)
print(a / c)
print(c ** a)
print(b - a)
print(a % c)
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Variáveis

Conversão de dados

- Caso seja necessário, é possível fazer conversão dos dados de alguma variável.
- Por exemplo:
  1. texto = "26"
  2. numero = int(texto)
  3. type(numero)
  4. int
- Ao fazer conversões, é preciso tomar cuidado. Por exemplo, se o valor da variável *texto* for "Mario", não será possível fazer a conversão de **str** para **int** e ocorrerá um erro de execução.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Variáveis

Conversão de dados: Exemplos

- Exemplo 1:
  1. numero = 0
  2. logico = bool(numero)
  3. print(logico)
  4. False
- Exemplo 2:
  1. numero = 1
  2. logico = bool(numero)
  3. print(logico)
  4. True
- Exemplo 3:
  1. inteiro = 1
  2. real = float(inteiro)
  3. print(real)
  4. 1.0

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Constantes

- Em Python, não existe uma definição de constante. Portanto, por convenção, se usa variáveis com letras maiúsculas para indicar que aquela variável é uma constante. Ou seja, que seu valor não deve ser alterado pelo programador.
- Exemplos:
  1. PI = 3.141592
  2. E = 2.718281
  3. print("Valor aproximado de pi: {}".format(PI))
  4. print("Valor aproximado de e: {}".format(E))

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Sequência de tempo

1. divida = 0
2. compra = 100
3. divida = divida + compra
4. compra = 200
5. divida = divida + compra
6. compra = 300
7. divida = divida + compra
8. compra = 0
9. **print**(divida)

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Entrada de dados

- A instrução **input** permite que um valor seja lido do teclado e armazenado em uma variável.
- Por exemplo:

```
nome = input("Digite o seu nome")
print("Seu nome é: {}".format(nome))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Entrada de dados

- O tipo do valor lido através da instrução **input** é string.
- Para podermos fazer cálculos com o valor recebido, deveremos usar as instruções de conversão.

```
a = int(input("Digite um número inteiro"))
b = int(input("Digite outro número inteiro"))
print(a + b)
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Exercícios

1. Faça um programa que peça dois números inteiros. Imprima a soma desses dois números na tela.
2. Escreva um programa que leia um valor em metros e exiba em milimetros.
3. Faça um programa que calcule o aumento de um salário. Ele deve solicitar o valor do salário e a porcentagem do aumento. Exiba o valor do aumento e do novo salário.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Tópicos

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Operadores relacionais

- É possível expressar sentenças que sejam falsas ou verdadeiras em Python.
- Para isso, utiliza-se operadores relacionais.

Operadores relacionais			
Símbolo	Significado	Exemplo	Resultado
$>$	maior que	$5 > 3$	True
$<$	menor que	$4 < 3$	False
$>=$	maior ou igual que	$10 >= 10$	True
$<=$	menor ou igual que	$7 <= 10$	True
$==$	igual a	$7 == 7$	True
$!=$	diferente de (não igual)	$7 != 7$	False

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Desvio condicional

Instrução **if**

- As condições servem para indicar quando uma parte do programa deve ser executada ou não.
- Em Python, a estrutura de decisão é o **if**.

if <condição>:
    Executa código caso a condição seja verdadeira.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Desvio condicional

Instrução if: Exemplo 1

- Este algoritmo pede como entrada dois números inteiros a e b.
- Ao terminar a execução, ele indica qual número digitado é maior.

```
a = int(input("Digite o primeiro valor"))
b = int(input("Digite o segundo valor"))
if a > b:
    print("O primeiro número é maior do que o segundo")
if b > a:
    print("O segundo número é maior do que o primeiro")
```

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Desvio condicional

Instrução if: Exemplo 2

- Este algoritmo pede um número inteiro como entrada e armazena o resultado na variável idade_carro.
- Ao terminar a execução, com base na idade, ele informa se o carro é novo ou velho.

```
idade_carro = int(input("Digite a idade do seu carro"))
if idade_carro <= 3:
    print("Seu carro é novo")
if idade_carro > 3:
    print("Seu carro é velho")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Desvio condicional

Instrução if: Exercício

- Escreva um programa que pergunte a velocidade do carro de um usuário. Caso ultrapasse 80km/h, exiba uma mensagem dizendo que o usuário foi multado. Nesse caso, exiba o valor da multa cobrando R$5,00 por km acima de 80km/h

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Desvio condicional

Instrução if...else

- Quando existem situações em que a segunda condição é o inverso da primeira (exemplo do carro), podemos usar a cláusula **else** para especificar o que fazer caso a *condição* do **if** for falsa. Sem precisar usar a instrução **if** novamente.
- Estrutura **if...else**

if <condição>:
    Executa código caso a condição seja verdadeira.
else:
    Executa código caso a condição seja falsa.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Desvio condicional

Instrução if...else: Exemplo do carro

- Exemplo do algoritmo do carro utilizando a estrutura **if...else**

```
idade_carro = int(input("Digite a idade do seu carro"))
if idade_carro <= 3:
    print("Seu carro é novo")
else:
    print("Seu carro é velho")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Desvio condicional

Instrução if...else: Exemplo média final

- Algoritmo que recebe dois números reais como entrada: *mp* e *avf*. Em seguida, a saída do algoritmo informa se o aluno foi aprovado ou reprovado.

```
mp = float(input("Digite a média ponderada"))
avf = float(input("Digite avaliação final"))
mf = (mp + avf)/2
if mf >= 5:
    print("Você foi aprovado")
else:
    print("Você foi reprovado")
```

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Estruturas aninhadas

1. Nem sempre os programas serão simples.
2. Muitas vezes, precisaremos aninhar vários **if** para obter o comportamento desejado do programa.
3. Aninhar, nesse caso, é utilizar um **if** dentro de outro.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Estruturas aninhadas

- Suponha que temos diferentes categorias de produto.
- Dependendo da categoria comprada, o preço do produto será diferente.
- Como fazer um algoritmo, com o conteúdo visto em sala, que leia a categoria de um produto e determine o preço dele?

Categoria	Produto
1	R$10,00
2	R$18,00
3	R$23,00

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Estruturas aninhadas

```
categoria = int(input("Digite a categoria do produto"))
if categoria == 1:
    preco = 10
else:
    if categoria == 2:
        preco = 18
    else:
        if categoria == 3:
            preco = 23
        else:
            print("Categoria inexistente")
            preco = 0
print("O preço do produto é: {}".format(preco))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Estruturas aninhadas

Instrução elif

- Solução anterior não é elegante.
- Coloca vários **if...else** dentro de outros.
- E se fosse uma tabela com vários elementos como no exemplo abaixo? O código teria vários espaçamentos para direita.
- Nesse caso, a melhor solução é usar a instrução **elif**.

Categoria	Produto
1	R$10,00
2	R$18,00
3	R$23,00
4	R$26,00
5	R$31,00

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Estruturas aninhadas

Instrução elif

```
categoria = int(input("Digite a categoria do produto"))
if categoria == 1:
    preco = 10
elif categoria == 2:
    preco = 18
elif categoria == 3:
    preco = 23
elif categoria == 4:
    preco = 26
elif categoria == 5:
    preco = 31
else:
    print("Categoria inválida")
    preco = 0
print("O preço do produto é: {}".format(preco))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Estruturas aninhadas

Instrução elif: Exemplos

- Escreva um programa que recebe um inteiro como entrada e produza os resultados "positivo", "negativo" ou "nulo" como saída.

```
n = int(input("Digite um número"))
if n > 0:
    print("Positivo")
elif n < 0:
    print("Negativo")
else:
    print("Nulo")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Estruturas aninhadas

Instrução elif: Exercícios

1. Escreva um programa que calcule o preço a pagar pelo fornecimento de energia elétrica. Pergunte a quantidade de kWh consumida e o tipo de instação: *R* para residências, *I* para indústrias e *C* para comércios. Calcule o preço a pagar de acordo com a tabela a seguir:

Preço por tipo de consumo		
Tipo	Faixa	Preço
Residencial	Até 500	R$0,40
	Acima de 500	R$0,65
Comercial	Até 1000	R$0,55
	Acima de 1000	R$0,60
Industrial	Até 5000	R$0,55
	Acima de 5000	R$0,60

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Operadores lógicos

- Assim como na lógica formal, boa parte das linguagens de programação utilizam operadores de negação, conjunção e disjunção.
- O uso adequado de operadores lógicos pode reduzir o número de instruções **if** no código.
- Em Python, se utilizada **not** para negação, **and** para conjunções e **or** para disjunções.

Operador	Operação
not	Negação
and	Conjunção
or	Disjunção

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Operadores lógicos

Conjunções

- Assim como na lógica formal, o operador **and** retorna verdadeiro apenas quando todas as condições mencionadas forem simultaneamente verdadeiras.

```
numero = int(input("Digite um valor inteiro"))
if numero >= 20 and numero <= 90:
    print("O valor está entre 20 e 90")
else:
    print("O valor não está entre 20 e 90")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Operadores lógicos

Conjunções

- Escreva um Programa que pergunte ao usuário que horas são (apenas a hora) e posteriormente mostre na tela "Bom dia", "Boa tarde" ou "Boa noite".

```
horas = int(input("Que horas são? [0-23]"))
if horas > 3 and horas < 12 :
    print("Bom dia")
elif horas >= 12 and horas < 18:
    print("Boa tarde")
else:
    print("Boa noite")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Operadores lógicos

Disjunções

- Assim como na lógica formal, o operador **or** retorna verdadeiro se pelo menos uma das condições mencionadas for verdadeira.

```
codigo = int(input("Entre com o código de acesso"))
if codigo == 1 or codigo == 2:
    if codigo == 1:
        print("O código digitado foi 1")
    else:
        print("O código digitado foi 2")
else:
    print("Código inválido")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Operadores lógicos

Disjunções

- Escreva um programa que verifique se um caractere digitado é ou não uma vogal.

```
c = input("Digite um caractere")
if c == 'a' or c == 'e' or c == 'i' or c == 'o' or c == 'u':
    print("É vogal")
else:
    print("Não é vogal")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Operadores lógicos

Negação

- Assim como na lógica formal, o operador **not** inverte o valor lógico de uma condição.

```
valor = int(input("Entre com um valor inteiro"))
if not valor >= 10:
    print("Valor informado: {}".format(valor))
else:
    print("Valor maior ou igual a 10. Inválido")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Tópicos

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Repetições

- Repetições são necessárias na maioria dos programas.
- São utilizadas para executar várias vezes a mesma sequência de instruções.
- Um exemplo simples que mostra a utilidade delas é: Como imprimir os 100 primeiros números naturais?

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Repetições

Imprimir os 100 primeiros números naturais

- Resolver esse problema para os 3 primeiros números é algo que já conseguimos fazer:

```
print(1)
print(2)
print(3)
```

- Outra solução:

```
x = 1
print(x)
x = x + 1
print(x)
x = x + 1
print(x)
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Repetições

Imprimir os 100 primeiros números naturais

- Usar a instrução **print** 100 vezes, apesar de resolver o problema, não parece ser uma solução muito elegante.
- Para resolver problemas desse gênero, as linguagens de programação utilizam **estruturas de repetição**.
- Uma das mais comuns é a instrução **while**.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Instrução **while**

- A instrução **while**, assim como a instrução **if**, checa uma condição.
- As instruções do bloco **while** são executadas repetidas vezes até a condição se tornar falsa.

```
# executa instruções enquanto <condição> é verdadeira
while <condição>:
    instrução 1
    instrução 2
    .
    .
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Instrução **while**

- Em geral, é comum usarmos expressões do tipo "x = x + 1", "x = x - 1" ou x = x + 5 em estruturas de repetição;
- Para tornar a escrita do código mais simples, Python permite expressar essas operações de uma maneira mais simplificada.

Expressão	**Forma compacta**
$x = x + y$	x += y
$x = x - y$	x -= y
$x = x * y$	x *= y
$x = x/y$	x /= y

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Imprimindo números naturais.

- Podemos aproveitar as estruturas de repetição para imprimir os 3 primeiros números naturais.
- 1ª repetição: testa 1<=3, imprime 1, atribui o valor 2 à i.
- 2ª repetição: testa 2<=3, imprime 2, atribui o valor 3 à i.
- 3ª repetição: testa 3<=3, imprime 3, atribui o valor 4 à i.

```
i = 1
while i <= 3:
    print(i)
    i = i + 1
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Imprimindo números naturais.

- A seguir, temos uma solução para o problema da impressão dos 100 primeiros números naturais:

```
contador = 1
while contador <= 100:
    print(contador)
    contador = contador + 1
```

- A solução a seguir, imprime números entre 50 e 100:

```
contador = 50
while contador <= 100:
    print(contador)
    contador += 1
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Exercício

1. Faça um algoritmo que imprima a contagem regressiva do lançamento de um foguete. O programa deverá imprimir 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 e Fogo! na tela. (Os números não precisam estar na mesma linha).

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Imprimindo números pares.

- Também é possível usar a instrução **while** em conjunto com a instrução **if**.
- Por exemplo, como imprimir os números pares que estão entre 1 e *n*? (considere *n* um valor digitado pelo usuário)

```
n = int(input("Digite um número maior do que 1"))
i = 1
while i <= n:
    if i % 2 == 0:
        print(i)
    i = i + 1
```

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Imprimindo números pares.

- Outra maneira de resolver o problema dos números pares entre 1 e *n*, é incrementar a variável *i* de dois em dois:

```
n = int(input("Digite um número maior do que 1"))
i = 2
while i <= n:
    print(i):
    i = i + 2
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Pontuação em questão.

- Pontuação de questões

```
pontos = 0
questao = 1
while questao <= 3:
    resposta = input("Resposta da questão {}".format(questao))
    if questao == 1 and resposta == 'b':
        pontos = pontos + 1
    if questao == 2 and resposta == 'a':
        pontos = pontos + 1
    if questao == 3 and resposta == 'd':
        pontos = pontos + 1
    questao += 1
print("O aluno fez {} pontos".format(pontos))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Exercícios

1. Faça um algoritmo que imprima os números ímpares entre 1 e n. Considere que n é um valor digitado pelo usuário.
2. Faça um algoritmo que receba um número entre 1 e 9 digitado pelo usuário. O algoritmo deve imprimir a tabuada desse número. Por exemplo, caso o usuário digite 2, a saída deverá ser: 2x1: 2, 2x2: 4, ..., 2x9: 18.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Acumulando resultados.

- Outro problema interessante é: Qual a soma dos 5 primeiros números naturais?

```
cont = 1
soma = 0
while cont <= 5:
    soma = soma + cont
    cont = cont + 1
print("Soma dos 5 primeiros números naturais: {}".format(soma))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução while

Acumulando resultados.

- Como calcular a média de 5 números digitados pelo usuário?

```
cont = 1
soma = 0
while cont <= 5:
    numero = float(input("Digite um número"))
    soma = soma + numero
    cont = cont + 1
print("Média dos números digitados: {}".format(soma/5))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Exercícios

1. Faça um algoritmo que calcule a multiplicação de 2 números **inteiros positivos** digitados pelo usuário. No algoritmo, apenas a operação de soma poderá ser utilizada.
2. Faça um algoritmo que calcule a multiplicação de 2 números inteiros digitados pelo usuário. No algoritmo, só poderão ser usadas as operações de soma, subtração e módulo.
3. Faça um algoritmo que calcule o fatorial de um número inteiro digitado pelo usuário.
4. Faça um algoritmo que imprima os 10 primeiros múltiplos de 3.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução for

- Na instrução **while**, estamos sempre incrementando uma variável contadora para testar a condição de encerramento do laço.
- Em alguns problemas, podemos usar a instrução **for** para não ter que incrementar explícitamente o valor da variável contadora.
- A instrução **for** incrementa automaticamente a variável contadora. A variável irá assumir valores entre *inicio* e *fim - 1*.

for *contador* **in** range(início, fim):
    instrução 1
    instrução 2
    .
    .

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução for

Exemplo:

- Para imprimir os 5 primeiros números naturais, podemos usar a seguinte instrução **for**:

```
for i in range(1, 6):
    print(i)
```

- É importante observar que nesse código, a variável **i** assume valores inteiros entre [1, 6). Ou seja, entre *início* e *fim - 1*.
- Logo, os números exibidos na tela serão 1, 2, 3, 4 e 5.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução for

Exemplo:

- O código abaixo imprime a soma dos 5 primeiros números inteiros.
- Nesse contexto, usamos a variável soma para calcular a soma dos números.

```
soma = 0
for i in range(1, 6):
    soma = soma + i
print(soma)
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução for

Impressão de números pares

- Assim como na instrução **while**, podemos imprimir os números pares usando a instrução *if*.

```
for i in range(1, 11):
    if i % 2 == 0:
        print(i)
print("Continuando o programa...")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução for

Impressão de números pares

- Por outro lado, também podemos incrementar a variável *i* de 2 em 2 na instrução **for**.
- Podemos passar um terceiro parâmetro para **range**, indicando o número que será incrementado na variável *i*.
- Por exemplo:

```
for i in range(2, 11, 2):
    print(i)
print("Continuando o programa...")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Exercícios

1. Faça um algoritmo que calcule o fatorial de um número inteiro digitado pelo usuário (Use a instrução **for**).
2. Faça um algoritmo que imprima os números ímpares entre 1 e n. Considere que n é um valor digitado pelo usuário (Use a instrução **for**).
3. Faça um algoritmo que receba um número entre 1 e 9 digitado pelo usuário. O algoritmo deve imprimir a tabuada desse número. Por exemplo, caso o usuário digite 2, a saída deverá ser: 2x1: 2, 2x2: 4, ..., 2x9: 18 (Use a instrução **for**).

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução break

- Apesar de raro, algumas vezes precisaremos terminar a execução de nossa estrutura de repetição antes da condição ser atendia.
- O código abaixo finaliza o laço quando a variável *cont* recebe o valor 5.

```
cont = 0
while cont <= 10:
    if cont == 5:
        break
    print(cont)
    cont += 1
print("Continuando o programa...")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução break

Instrução for

- Assim como na instrução **while**, também é possível usar o **break** na instrução **for**.

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
print("Continuando o programa...")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução continue

Instrução while

- Outras vezes, precisamos que a estrutura de repetição não execute em determinados casos.
- Exemplo: Como imprimir todos números inteiros entre 1 e 10 que não sejam 5 e 9?
- Uma das soluções, é com a instrução **if**.

```
cont = 0
while cont < 10:
    cont += 1
    if cont != 5 and cont != 9:
        print(cont)
print("Continuando o programa...")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução continue

Instrução while

- Outra solução, é através da instrução **continue**

```
cont = 0
while cont < 10:
    cont += 1
    if cont == 5 or cont == 9:
        continue
    print(cont)
print("Continuando o programa...")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Instrução continue

Instrução for

- Assim como na instrução **while**, também é possível usar o **continue** na instrução **for**.

```
for i in range(1, 11):
    if i == 5 or i == 9:
        continue
    print(i)
print("Continuando o programa...")
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Tópicos

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Funções

- No decorrer do curso, utilizamos métodos para imprimir na tela e receber dados do usuário utilizando os comandos **print** e **input**.
- Também utilizamos os comandos **int** e **float**, para converter strings digitadas pelo usuário em números inteiros ou reais.
- Esses comandos, são funções/procedimentos já existentes em Python.
- Em geral, as linguagens de programação permitem que os programadores criem suas próprias funções e procedimentos.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Funções

- Funções e procedimentos podem ser vistos como um conjunto de instruções que resolvem um problema bem definido. Por exemplo:
  1. Função/procedimento que verifica se um número é par ou impar.
  2. Função/procedimento que soma 3 números.
- Eles podem ser criados com duas finalidades:
  1. Reaproveitar código (não codificar a mesma coisa várias vezes!).
  2. Dividir um problema complexo em problemas menores (Resolvendo cada problema menor separadamente).

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Funções

- Em uma função, após resolver o problema definido, obrigatoriamente se retorna um valor.
- Em Python, a função é definida através da instrução **def** e finalizada ao usar a instrução **return**. Por exemplo:

```
def nome_funcao(parametro_1, parametro_2, ..., parametro_n)
    instrução 1
    instrução 2
    .
    .
    .
    return valor
```

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Funções

Exemplos

- Exemplos:

```
def soma(a, b):
    resultado = a + b
    return resultado
```

```
def subtracao(a, b):
    resultado = a - b
    return resultado
```

```
def pi():
    return 3.14159265359
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Funções

Exemplos

- As funções precisam ser definidas **antes** de nosso programa principal.
- Após a definição das funções, poderemos utiliza-las em nosso código.
- Outro aspecto importante, é que o valor retornado pela função poderá ser utilizado pelo programador. Seja para atribuir esse valor à uma variável, imprimi-lo na tela ou usar como entrada de outra função.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Funções

Exemplos

- Exemplo:

```
def soma(a, b):
    resultado = a + b
    return resultado
variavel_a = soma(5, 3)
variavel_b = soma(soma(3,5), 3)
print(variavel_a)
print(variavel_b)
print(soma(1,3))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Procedimentos

- Um procedimento apenas resolve um problema bem definido. Nele, não se retorna valores.
- Em Python, um procedimento é definido da mesma forma que uma função. Contudo, a instrução **return** não é usada no final.

```
def nome_funcao(parametro_1, parametro_2, ..., parametro_n)
    instrução 1
    instrução 2
    .
    .
    instrução n
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Procedimentos

Exemplos

```
def imprima(texto):
    print(texto)
```

```
def linha():
    print("**********")
```

```
def imprime_soma(a, b):
    print(a + b)
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dividir para conquistar

- Suponha que precisamos resolver o seguinte problema: Somar o fatorial dos números primos entre 1 e 10.
- Para resolver esse problema, podemos dividi-lo em etapas:
  1. Encontrar os números primos entre 1 e 10.
  2. Calcular o fatorial desses números.
  3. Realizar a soma dos fatoriais.
- Esse problema se tornaria elementar se tivessemos uma função que calcula o fatorial de um número e uma função que me diz se um número é primo.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dividir para conquistar

Funções do problema

```
def fatorial(n):
    resultado = 1
    for i in range(1, n+1):
        resultado = resultado * i
    return resultado
def primo(n):
    resultado = False
    contador = 0
    for i in range(1, n+1):
        if n % i == 0
            contador = contador + 1
    if contador == 2:
        resultado = True
    return resultado
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dividir para conquistar

Solução do problema

- Agora que dividimos o problema em partes menores usando funções, podemos facilmente obter uma solução:

```
soma = 0
for i in range(1, 11):
    if primo(i):
        soma = soma + fatorial(i)
print(soma)
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

- Geralmente nos deparamos com problemas que são comuns (no sentido dele aparecer em várias situações).
- Algumas vezes, já resolvemos anteriormente um problema que nos aparece atualmente.
- Outras vezes, outras pessoas já resolveram o problema que está diante de nós.
- Nessas situações, será que precisamos "reinventar a roda"?

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

Reaproveitando nossos códigos

- Podemos salvar nossas funções em arquivos com a extenção .py.
- Ao criarmos um novo algoritmo, se ele estiver na mesma pasta que o arquivo de nossas funções, poderemos reaproveitar essas funções.
- Para isso, usaremos a instrução **import**.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

Reaproveitando nossos códigos

- Suponha que as seguintes funções estejam salvas em um arquivo operacoes.py

```
def soma(a, b):
    return a + b
def sub(a, b):
    return a - b
def mult(a, b):
    return a * b
def div(a, b):
    return a / b
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

Reaproveitando nossos códigos

- Agora que já criamos as funções, podemos utilizá-las sem precisar codificar elas novamente.

```
import operacoes
print(operacoes.soma(2,5))
print(operacoes.div(10,5))
print(operacoes.sub(2,5))
print(operacoes.mult(10,4))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

Reaproveitando nossos códigos

- Para não ter que digitar 'operacoes' toda vez que for usar uma função, podemos usar a instrução **as** para simplificar o código.

```
import operacoes as op
print(op.soma(2,5))
print(op.div(10,5))
print(op.sub(2,5))
print(op.mult(10,4))
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

Reaproveitando nossos códigos

- Também podemos importar funções especificamente. Por exemplo:

```
from operacoes import soma, div
print(soma(2,5))
print(div(10,5))
```

- Nesse caso, apenas as funções div e soma foram importadas. Ocorrerá um erro de execução se tentarmos usar as outras funções.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

Reaproveitando nossos códigos

- Finalmente, também é possível importar todas as funções ao usar o *

```
from operacoes import *
print(soma(2,5))
print(sub(5,2))
print(mult(10,3))
print(div(10,5))
```

- Em grandes projetos, esse import não é recomendável. As chances de existirem funções com o mesmo nome em diferentes arquivos é alta.

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

- Assim como podemos reaproveitar código que nós escrevemos, também podemos aproveitar código de outras pessoas!
- Por exemplo, podemos utilizar o conjunto de funções de **math** para realizar operações matemáticas mais complexas.
- Basta usar **import** math.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Reaproveitamento de código

- Alguns exemplos de funções que estão na biblioteca de funções **math** são:
  1. math.**sin**(x): Retorna o seno de x.
  2. math.**cos**(x): Retorna o cosseno de x.
  3. math.**tan**(x): Retorna a tangente de x.
  4. math.**pow**(x, y): Retorna x^y.
  5. math.**factorial**(x): Retorna o fatorial de x.

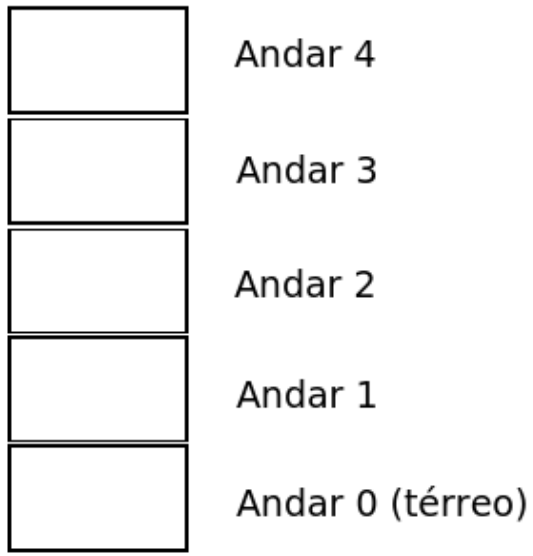
UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Tópicos

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

- Em Python, o tipo lista é uma estrutura de dados que permite o armazenamento de vários valores que são acessados por um índice.
- O tamanho de uma lista é igual à quantidade de elementos que ela contém.
- Podemos imaginar uma lista como um edifício de apartamentos.



UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

Criando e acessando elementos de listas

- Exemplos de como criar uma variável do tipo lista:
  - a = [1, 2, 3, 4]
  - b = [1.5, "casa", 3, False, 10]
  - c = ["a", "b", "c", "d"]
  - d = [ ]
- Como acessar elementos de uma lista:
  - print(a[0]) # imprime 1
  - print(c[2]) # imprime c
  - print(b[3]) # imprime False

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

Alterando valores da lista

- Também podemos usar o operador de atribuição para modificar elementos da lista.

```
notas = [10, 8, 4.5, 7.6, 6]
print(notas)
# muda a nota da posição 0
notas[0] = 2.5
# muda a nota da posição 2
notas[2] = 10
print(notas)
>> [2.5, 8, 10, 7.6, 6]
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

Acesso aos elementos com estruturas de repetição

- A seguir, um exemplo de como acessar os elementos utilizando a instrução **for**.

```
notas = [9, 8, 4.5, 7.6, 6]
tamanho = 5
for index in range(0, tamanho):
    print(notas[index])
```

- Como dar um ponto a mais para cada nota?

```
for index in range(0, tamanho):
    notas[index] = notas[index] + 1
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

Funções úteis para manipular listas

- Existem funções que podemos usar para fazer manipulações mais elaboradas com as listas:
  1. **len**()
  2. **append**()
  3. **pop**()

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

Função **len**

- A função **len** retorna o número de elementos da lista. Por exemplo:

```
notas = [9, 8, 4.5, 7.6, 6]
nomes = ["Ash", "Brocky", "Misty"]
idades = [3, 15, 25, 28, 14, 35, 65]
vazio = [ ]
print(len(notas)) # imprime 5
print(len(nomes)) # imprime 3
print(len(idades)) # imprime 7
print(len(vazio)) # imprime 0
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

Procedimento **append()**

- O procedimento **append** adiciona um elemento ao final da lista. Por exemplo:

```
notas = [ ]
notas.append(8) # [8]
notas.append(5.5) # [8, 5.5]
notas.append(7.6) # [8, 5.5, 7.6]
notas.append(9.6) # [8, 5.5, 7.6, 9.6]
print(notas)
>> [8, 5.5, 7.6, 9.6]
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Listas

Função pop()

- A função **pop** recebe um índice como parâmetro, remove o elemento que está na posição do índice e retorna o valor removido. Por exemplo:

```
notas = [8, 5.5, 7.6, 9.6]
# remove a nota na posição 1 e armazena na variável a
a = notas.pop(1)
print(notas)
>> [8, 7.6, 9.6]
print(a)
>> 5.5
# remove a nota na posição 0 e armazena na variável b
b = notas.pop(0)
print(notas)
>> [7.6, 9.6]
```

UFCA UNIVERSIDADE FEDERAL DO CARIRI

Listas

Exercícios

- Faça uma função que tenha uma lista de números como parâmetro. Ela deve retornar a soma dos elementos da lista.
- Faça uma função que tenha uma lista de números como parâmetro. Ela deve retornar a média dos elementos da lista.
- Faça uma função que tenha duas listas (**a** e **b**) de números como parâmetro. A função deve retornar uma nova lista **l** de tal forma que **l**[i] = **a**[i] + **b**[i] para qualquer i entre 0 e **len(a)**. PS: O tamanho de **a** e **b** devem ser iguais!.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dicionários

- Em Python, um dicionário é uma estrutura de dados que associa os elementos à chaves.
- Diferente das listas, que associam os elementos à índices.

```
estoque = {'tomate': 1000, 'alface': 500, 'batata': 2000}
print(estoque['tomate'])
>> 1000
print(estoque['alface'])
>> 500
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dicionários

Criando e modificando elementos

- Podemos utilizar o operador de atribuição para criar novos elementos ou modificar elementos existentes. Por exemplo:

```
estoque = {'tomate': 1000, 'alface': 500}
estoque['tomate'] = 1
print(estoque)
>> {'tomate': 1, 'alface': 500}
estoque['tangerina'] = 300
print(estoque)
>> {'tomate': 1, 'alface': 500, 'tangerina': 300}
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dicionários

Criando e modificando elementos

- Assim como nas listas, também podemos utilizar a função **pop**() nos dicionários.

```
estoque = {'tomate': 1000, 'alface': 500}
a = estoque.pop('alface')
print(estoque)
>> {'tomate': 1000 }
print(a)
>> 500
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dicionários

Pecorrendo os elementos de um dicionário

- A variável *chave* na instrução **for** abaixo, assume o valor de uma chave do dicionário a cada iteração.

```
estoque = {'tomate': 1000, 'alface': 500, 'batata': 400 }
for chave in estoque.keys():
    print("chave: {}; valor: {}".format(chave, estoque[chave]))
>> chave: tomate; valor: 1000
>> chave: alface; valor: 500
>> chave: batata; valor: 400
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Dicionários

Pecorrendo os elementos de um dicionário

- Dicionários também podem ser utilizados com listas.
- Esse tipo de estrutura é bastante útil para representar diversas situações.

```
notas = {'Naruto': [2.3, 1.5, 4.5],
         'Sasuke': [10, 9.6, 9.8],
         'Sakura': [7.2, 6.8, 8.0] }
faltas = {'Naruto': [0, 0, 0, 1, 0, 1, 1, 0, 0, 0],
          'Sasuke': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          'Sakura': [0, 0, 0, 0, 0, 1, 0, 0, 0, 1] }
```

UFCA
UNIVERSIDADE FEDERAL DO CARIRI

Exercícios

- Crie um dicionário em que as chaves são nomes de alunos e os elementos são uma lista de notas. Calcule e mostre a média de cada aluno.
- Crie um dicionário em que as chaves são nomes de alunos e os elementos são uma lista de frequencias. Calcule a porcentagem de faltas de cada aluno e diga se o aluno foi ou não reprovado por falta.

UFCA
UNIVERSIDADE FEDERAL DO CARIRI