MAC 115 — Introdução à Ciência da Computação

Aula 14

Nelson Lago

IF noturno – 2023


E a prova, hein?

Prova – questão 1

Simule o código abaixo e selecione as opções correspondentes à saída (S1 a S5) impressa do programa.

Prova – questão 1

Simule o código abaixo e selecione as opções correspondentes à saída (S1 a S5) impressa do programa.

```
n = 42
a = 2
while n >= a:
    b = n // 2
    if b * 2 == n:
        a = a + 3
    else:
        a = a + 2
    n = n - 7
    x = a % 9
    print(x * 7 + 12)
```

Prova – questão 2

Escreva um programa em Python responsável por controlar um caixa eletrônico. O programa deve perguntar ao usuário o valor do saque desejado e fornecer um conjunto de notas correspondente a esse valor. O caixa eletrônico não dispõe de notas menores que R$10 e não permite saques de mais de R$1.000. Além disso, seu sistema deve evitar fornecer notas de R$100; elas devem ser utilizadas apenas se o número de notas de R$50 for maior que 4.

Prova – questão 2

```
def main():

    while not valido:


            print("O maior saque permitido é R$1.000")



    falta = solicitado



        notasCem += 1
        notasCinquenta -= 2

    print("Você vai receber R${} com {} notas de 100, "
          "{} notas de 50 e {} notas de dez".format(solicitado, notasCem, notasCinquenta, notasDez))
```

Prova – questão 2

```
def main():
    valido = False
    while not valido:
        solicitado = int(input("Digite o valor do saque: "))
        if solicitado > 1000:
            print("O maior saque permitido é R$1.000")
        elif solicitado % 10 != 0:
            print("A menor nota disponível é R$10")
        else:
            valido = True
    falta = solicitado
    notasCinquenta = falta // 50
    falta %= 50
    notasCem = 0
    while notasCinquenta > 4:
        notasCem += 1
        notasCinquenta -= 2
    notasDez = falta // 10
    print("Você vai receber R${} com {} notas de 100, "
          "{} notas de 50 e {} notas de dez".format(solicitado, notasCem, notasCinquenta, notasDez))
```

Prova – questão 3

Escreva um programa em Python, completando as lacunas, que informa quantas horas uma pessoa viveu entre sua data e hora de nascimento e uma data e hora fornecidas pelo usuário. Cada conjunto de data e horário é fornecido por quatro valores inteiros, de forma que o primeiro representa o dia, o segundo o mês, terceiro o ano e o quarto a hora (sem minutos e segundos). A segunda data fornecida é sempre cronologicamente maior que a primeira.

Prova – questão 3

```
def bissexto(a):

def TotalDiasPorMes(m, a):

        ndias = 29

        ndias = 28

        ndias = 30
    else:
        ndias = 31


    d1,m1,a1,h1 = int(input("Dia: ")),int(input("Mês: ")),int(input("Ano: ")),int(input("Hora: "))
    d2,m2,a2,h2 = int(input("Dia: ")),int(input("Mês: ")),int(input("Ano: ")),int(input("Hora: "))
    horas = 0
    d, m, a, h = d1, m1, a1, h1

        horas, d = horas +24, d +1


        if m > 12:


    print("Quantidade de horas:", horas)
```

Prova – questão 3

```
def bissexto(a):
    return (a % 4 == 0 and a % 100 != 0) or a % 400 == 0
def TotalDiasPorMes(m, a):
    if bissexto(a) and m == 2:
        ndias = 29
    elif m == 2:
        ndias = 28
    elif (m % 2 == 0 and m < 7) or (m % 2 != 0 and m > 8):
        ndias = 30
    else:
        ndias = 31
    return ndias
def main():
    d1,m1,a1,h1 = int(input("Dia: ")),int(input("Mês: ")),int(input("Ano: ")),int(input("Hora: "))
    d2,m2,a2,h2 = int(input("Dia: ")),int(input("Mês: ")),int(input("Ano: ")),int(input("Hora: "))
    horas = 0
    d, m, a, h = d1, m1, a1, h1
    while d<d2 or m<m2 or a<a2:
        horas, d = horas +24, d +1
        if d > TotalDiasPorMes(m,a):
            d, m = 1, m+1
        if m > 12:
            m, a = 1, a+1
    horas += h2 - h1
    print("Quantidade de horas:", horas)
```

Prova – questão 4

Escreva um programa Python que calcule o valor do arco-tangente de m tangentes de ângulos dadas x_i, $i = 0, \ldots m$, usando os n primeiros termos da aproximação abaixo (série de Taylor):

$$arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

Além de calcular a aproximação pela série acima, seu programa deve calcular o erro absoluto da aproximação usando a função `atan` do módulo `math` do Python. Seu programa deve permitir a realização de m testes pelo usuário, ou seja, a leitura de m tangentes x de arcos distintos.

Prova – questão 4

```
import math
def main():
  m = int(input("testes:")); n = int(input("termos:"))
```

```
    print(esperado, estimado, erro)
main()
```

Prova – questão 4

```
import math
def main():
  m = int(input("testes:")); n = int(input("termos:"))
  i = 0
  while i < m:
    estimado = 0.0
    x = float(input('tangente do arco (-1 < x < 1):'))
    sinal = -1
    j = 0
    while j < n:
      sinal = -sinal
      fator = 2*j +1
      termo = (x**fator) / fator
      estimado += sinal * termo
      j += 1
    esperado = math.atan(x)
    erro = abs(esperado - estimado)
    i += 1
    print(esperado, estimado, erro)
main()
```

And now for something completely different

Oncotô?

Nós já podemos:

Oncotô?

Nós já podemos:

- **Interagir com o “mundo”**

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`
  - Apresentar dados — `print()`

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`
  - Apresentar dados — `print()`
    - » *Lembrando que dados podem ter **tipos** diferentes*

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`
  - Apresentar dados — `print()`
    - » *Lembrando que dados podem ter **tipos** diferentes*
- **Guardar dados com um *nome* — variáveis**

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`
  - Apresentar dados — `print()`
    - » *Lembrando que dados podem ter **tipos** diferentes*
- **Guardar dados com um *nome* — variáveis**
- **Processá-los — expressões como `+`, `-`, `format()` etc**

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`
  - Apresentar dados — `print()`
    - » *Lembrando que dados podem ter* ***tipos*** *diferentes*
- **Guardar dados com um *nome* — variáveis**
- **Processá-los — expressões como `+`, `-`, `format()` etc**
  - Que podem ser *compostas* — (5 + 7) / 2

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`
  - Apresentar dados — `print()`
    - » *Lembrando que dados podem ter* ***tipos*** *diferentes*
- **Guardar dados com um *nome* — variáveis**
- **Processá-los — expressões como `+`, `-`, `format()` etc**
  - Que podem ser *compostas* — (5 + 7) / 2
  - Tomando "decisões" — `if`

Oncotô?

Nós já podemos:

- **Interagir com o "mundo"**
  - *Ler* dados — `input()`
  - Apresentar dados — `print()`
    - » *Lembrando que dados podem ter* ***tipos*** *diferentes*
- **Guardar dados com um *nome* — variáveis**
- **Processá-los — expressões como `+`, `-`, `format()` etc**
  - Que podem ser *compostas* — (5 + 7) / 2
  - Tomando "decisões" — **`if`**
  - Fazendo repetições — **`while`**

Oncotô?

Também podemos:

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis “dentro” de cada uma normalmente não são visíveis para as outras — escopo

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

```
def média(a, b):
    return (a + b) / 2
```

Oncotô?

Também podemos:

- **Dividir o programa em partes com *nomes* — funções**
  - Cada parte funciona de maneira independente, pois as variáveis "dentro" de cada uma normalmente não são visíveis para as outras — escopo
- **Essas partes também podem interagir com o "mundo"**
  - `input()` e `print()`
- **Ou podem interagir com as outras partes do programa**
  - *recebendo* e *devolvendo* dados — parâmetros e **`return`**

```
def média(a, b):
    return (a + b) / 2
```

  - Modificando uma variável global — **`global`**

O que está faltando?

O que está faltando?

Coleções

Coleções

Coleções

- **Lista de clientes**

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Faz sentido inventar nomes (variáveis) para cada um deles?

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Faz sentido inventar nomes (variáveis) para cada um deles? (dica: não 😂)

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Faz sentido inventar nomes (variáveis) para cada um deles? (dica: não 😂)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Faz sentido inventar nomes (variáveis) para cada um deles? (dica: não 😂)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- **Podemos manipular o conjunto como um todo**

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Faz sentido inventar nomes (variáveis) para cada um deles? (dica: não 😂)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- **Podemos manipular o conjunto como um todo**
- **Podemos manipular subconjuntos**

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Faz sentido inventar nomes (variáveis) para cada um deles? (dica: não 😂)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- **Podemos manipular o conjunto como um todo**
- **Podemos manipular subconjuntos**
- **Podemos manipular elementos um por um**

Coleções

- **Lista de clientes**
  - E, por que não, “sub-listas”, como a lista dos clientes que atendem a um dado critério (idade, cidade em que moram etc.)
- **Lista dos divisores de um número**
- **...**

Faz sentido inventar nomes (variáveis) para cada um deles? (dica: não 😂)

O que faz sentido é tratá-los como um *conjunto* que tem um nome

- **Podemos manipular o conjunto como um todo**
- **Podemos manipular subconjuntos**
- **Podemos manipular elementos um por um**
- **...**

Coleções

A principal coleção em python é a *lista*:

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
```

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
```

vermelho

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
print(cores[2])
```

---

vermelho

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
print(cores[2])
```

---

vermelho

amarelo

Por que computação?

O computador é extremamente rápido, mas é uma ferramenta com o mesmo nível de "inteligência" que um martelo

Não é mais fácil fazer manualmente?

Repetições e condições

- **"Algo" precisa acontecer para indicar que as repetições chegaram ao fim**

  (ok, às vezes queremos repetir indefinidamente, mas vamos ignorar isso por enquanto)

- **As repetições são controladas por algum tipo de *condição* baseada no estado de uma *variável***

Partes mínimas de um laço

- **Um laço correto precisa**
  - Inicializar a variável de controle antes do início do laço
  - Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
  - Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
for p in primos:
  print("O número", p, "é primo")
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
for p in primos:
  print("O número", p, "é primo")
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
for p in primos:
  print("O número", p, "é primo")
```

- **Um laço correto precisa**
  - Inicializar a variável de controle antes do início do laço
  - Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
  - Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O {}o primo é {}".format(n+1, primos[n]))
  n += 1
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O {}o primo é {}".format(n+1, primos[n]))
  n += 1
```

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 1
for p in primos:
  print("O {}o primo é {}".format(n, p))
  n += 1
```

Coleções

Dicas para o DJ:

Coleções

Dicas para o DJ:

Coleções

Dicas para o DJ:

```
sugestão = "nenhuma"
while sugestão != "":
  sugestão = input("Sugira uma canção para o DJ: ")
```

Coleções

Dicas para o DJ:

```
canções = []
sugestão = "nenhuma"
while sugestão != "":
  sugestão = input("Sugira uma canção para o DJ: ")
```

Coleções

Dicas para o DJ:

```
canções = []
sugestão = "nenhuma"
while sugestão != "":
  sugestão = input("Sugira uma canção para o DJ: ")
  if sugestão != "":
    canções.append(sugestão)
```

Coleções

Dicas para o DJ:

```
canções = []
sugestão = "nenhuma"
while sugestão != "":
  sugestão = input("Sugira uma canção para o DJ: ")
  if sugestão != "":
    canções.append(sugestão)
print("No total, foram sugeridas {} canções:".format(len(canções)))
print(canções)
```

Coleções

Dicas para o DJ:

```
canções = []
sugestão = "nenhuma"
while sugestão != "":
  sugestão = input("Sugira uma canção para o DJ: ")
  if sugestão != "":
    canções.append(sugestão)
print("No total, foram sugeridas {} canções:".format(len(canções)))
n = 0
while n < len(canções):
  print(canções[n])
  n += 1
```

Coleções

Dicas para o DJ:

```
canções = []
sugestão = "nenhuma"
while sugestão != "":
  sugestão = input("Sugira uma canção para o DJ: ")
  if sugestão != "":
    canções.append(sugestão)
print("No total, foram sugeridas {} canções:".format(len(canções)))
for canção in canções:
  print(canção)
```