DATA 16 03 22

Anotações de Aula de Mariana V.

Aulas 01 a 20
Semanas 01 - 07

Algoritmos e Programação de Computadores I

## Aula 01 Noção de Algoritmos

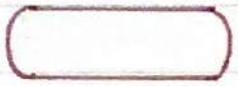
- Construções de Problemas
  - Definição (o que) → Desenvolvimento (como)
    - ↳ Projetar a solução (algoritmo)
    - ↳ Codificar a solução (programa)
    - ↳ Testar o programa

- Algoritmo
  - Sequência de passos que visam atingir um objetivo bem definido (deve indicar o início)
  - Qualidades: 1. Cada ação do algoritmo deve ser uma instrução que possa ser realizada
    2. Determinar ordem dos passos
    3. Deve ter um fim
  - Sequenciamento: estabelece um padrão de comportamento. Ações realizadas em sequencia, uma após a outra
  - Teste seletivo: determina qual conjunto de ações deve ser seguido, dependendo da condição resultar em verdadeiro ou falso.
  - Repetição: mesmo trecho é repetido várias vezes até que a condição de parada seja alcançada
    ↳ número de repetições é indefinido, porém finito.

DATA 16 03 22

## Aula 02 Fluxogramas

- Fluxograma
  - Representação esquemática de um algoritmo, através de gráficos que ilustram a transição de informações entre os elementos que o compõe
  - Simbologia:



Início ou fim do fluxograma

Entrada ou saída de dados

Instrução

Ponto de decisão



Fluxo de dados e conexão

## Aula 03 Sobre Python

- Python
  - Linguagem de programação de alto nível projetada para desenvolver programas de alta legibilidade
  - Desenvolvida na década de 80 por Guido van Rossum na Holanda

Interface Online:

jupyter.org

↳ Jupyter Notebook

- **IDE**
  - Integrated Development Environment
  - Aplicativo que fornece funcionalidades para o programador de software
  - Editor de texto, compilador e debugador de código

## Aula 04 — Expressões aritméticas e operadores

- **Operadores aritméticos**
  - `+` soma
  - `-` subtração
  - `*` multiplicação
  - `/` divisão
  - `//` divisão inteira, desconsidera o resto (o quociente da divisão sem o resto)
  - `%` retorna o resto da divisão
  - `**` potência
  - `( )` preferência de cálculo (mesmo uso da matemática)

- **Tipos de dados**
  - `int` números inteiros
  - `float` números com casas decimais

> - soma/subtração/multip. de dois int → resultado int
> - ao menos um float na expressão → resultado float
> - divisão de dois int → resultado float

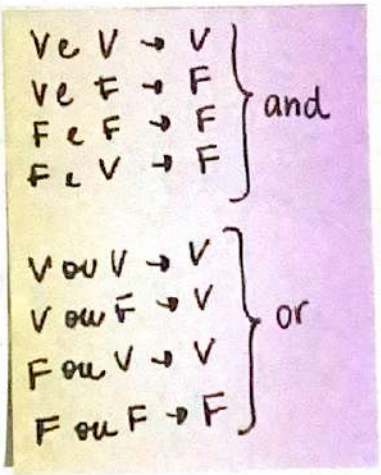
16 03 22

- Funções Matemáticas
  - abs() retorna o valor absoluto de um número
  - min() retorna o mínimo de um conjunto de dados
  - max() retorna o máximo de um conjunto de dados

## Aula 05 Expressões lógicas e operadores

- Operadores relacionais
  - retorna: verdadeiro ou falso
  - == igual a
  - != diferente de
  - > maior que
  - < menor que
  - >= maior ou igual
  - <= menor ou igual

- Operadores lógicos
  - and produto lógico
  - or soma lógica
  - not negação
  - in pertence (relacionado com uma lista)
  - not in não pertence
  - is elemento é de um tipo
  - not is elemento não é de um tipo




- Precedência de Operações
  1. Parênteses mais internos
  2. Operadores aritméticos
  3. Operadores relacionais
  4. Operadores lógicos (not → or and)

DATA 21 03 22

## Aula 06 Variáveis

- variáveis

- É um nome que é atribuído a um objeto (um número por exemplo).

<variável> = <expressão>

ex: >>> x = 3 + 3

- Nome de variáveis pode conter: (minúsculas e maiúsculas são variáveis diferentes)
  - caracteres em minúsculo (a-z)
  - caracteres em maiúsculo (A-Z)
  - underscore (_)
  - dígitos (0-9) exceto para 1º caractere

ex: myList ; _list ; list6 ; 1_6

+ Convenção para os nomes: (preço ao invés de p)
  - ↳ usar nomes significativos
  - ↳ mais de uma palavra: usar separador underscore ou capitalizer ex tempVar
  - ↳ nomes mais curtos são melhores

**Palavras Reservadas** (não podem ser utilizadas como variáveis no python)

| | | | |
|---|---|---|---|
| • False | • Break | • Else | • If |
| • None | • Class | • Except | • Import |
| • True | • Continue | • Finally | • In |
| • And | • Def | • For | • Is |
| • As | • Del | • From | • Lambda |
| • Assert | • Elif | • Global | • Nonlocal |
| • Not | • Pass | • Try | • With |
| • Or | • Raise | • While | • Yield |
| | • Return | | |

CICERO

DATA 21 03 22

# Aula 07 Strings

- String (str)

• Usado para representar e manipular texto ou uma sequência de caracteres, incluindo espaço em branco, pontuação e símbolos diferentes.

↳ é criada como uma sequência de caracteres envolvida por aspas (simples ou dupla).

ex: >>> a = 'Algoritmos e Programação'

- Operadores com strings

```
>>> s = 'abc'
>>> s == 'abc'
```
] True

```
>>> t = 'def'
>>> s < t
```
] True — utiliza ordem alfabética pl relacionar

```
>>> s+t
'abcdef'
>>> s*t
```
] Erro * não consegue usar multiplicador entre dois str

```
>>> s*2
'abcabc'
>>> ch = 'b'
>>> ch in s
```
] True

```
>>> ch is str('b')
```
] True

**Exemplo**

```
>>> s = 'abcd'
        0123
       -4-3-2-1

>>> len(s)
4
>>> s[0]
'a'
>>> s[-1]
'd'
>>> s[0:2]
'ab'
>>> s[-4,-2]
'ab'
>>> s[:3]
'abc'
```

• Operadores que <u>não funcionam</u> com strings:

-, /, //, %, **

`len()` retorna o tamanho da string (p número de caracteres)

`[ ]` operador de indexação (retorna o caractere da posição)

DATA 21 03 22

- Métodos para manipulação de Strings

| | |
|---|---|
| s.find(p) | retorna o índice em que a substring p aparece em s |
| s.count(p) | retorna a frequência em que a substring p aparece em s |
| s.replace(p, q) | substitui a substring q em s |
| s.capitalize() | substitui o primeiro caractere de s em maiúscula |
| s.upper() | substitui todos os caracteres de s em maiúscula |
| s.lower() | substitui todos os caracteres de s em minúscula |
| s.strip() | remove os espaços em branco em excesso |

## Aula 08 Listas, tuplas e operadores

- Listas

- É uma sequência de objetos
  - ↳ podem ser de diferentes tipos: números, Strings, outras listas

É reprensentada por objetos separados por virgulas, envoltos por colchetes []

- Operadores com listas

- Alguns operadores utilizados com os strings podem ser utilizados com listas:

in, not in, +, *, len(), []

min(), max(), sum()

DATA 23 03 22

- Listas: mutáveis

Diferença entre listas e strings:
↳ listas são mutáveis, isto é, seu conteúdo pode variar

EXEMPLO:
```
>>> pets = ['cão', 'gato', 'boi']
>>> pets [2] = 'vaca'
>>> pets
>>> pets
    ['cão', 'gato', 'vaca']
```

- Tuplas

Tuplas são iguais as listas, mas imutáveis, usa-se parênteses () para criá-los
ex: dias = ('seg', 'ter', 'qua')

- métodos para listas

lista.append(novo elemento) anexa um novo elemento no final da lista
lista.insert(pos, novo elemento) insere o novo elemento na posição da lista, caso essa não exista, será criada
lista.count(elemento) retorna quantas vezes o elemento aparece na lista
lista.sort(lista) ordena o conteúdo da lista
lista.pop() retorna e remove o último elemento da lista
lista.pop(pos) retorna e remove o elemento na pos
lista.remove(x) remove a primeira ocorrência do item x
lista.reverse() inverte a ordem dos elementos
lista.index() retorna a posição do elemento

DATA 24 03 22

## Aula 09 Tipos de Dados

- Tipos de dados
  - Cada dado é armazenado na memória na forma de objetos
    - todo objeto possui um tipo e valor
- Tipo de um objeto
  - Indica que tipo de valores o objeto pode manter e que tipo de operações podem ser realizadas sobre esse objeto.
  - Tipos já vistos:
    - `int` inteiro 3
    - `float` ponto flutuante 3.0
    - `bool` Booleano True/False
    - `str` String 'Hello World'
    - `lst` lista [1, 1, 2, 4, 5]
  - Determinar a função de um objeto: `type ( )`
    - variável não tem um tipo, apenas o objeto a que ela se refere tem um tipo
- Construtores
  - São funções utilizadas para instanciar explicitamente um objeto
    - `int ( )`
    - `float ( )`
    - `lst ( )`
    - `str ( )`

DATA 24 03 22

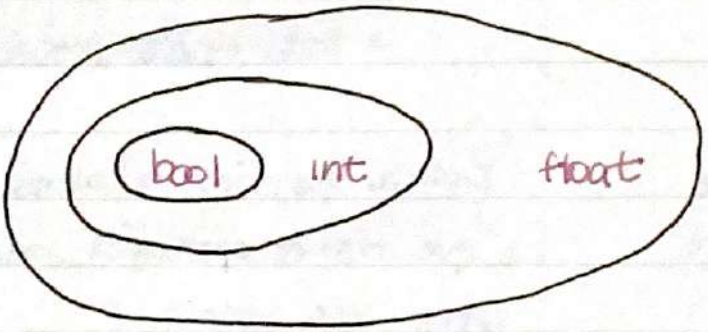
• Conversão de Tipo Implícita

Se uma operação algebrica ou lógica envolver operandos de diferentes tipos, o Python converterá cada operando para o tipo que contém os outros

**EXEMPLO:**

```
>>> True + 5
6

>>> 3 + 0.35
3.35
```




* bool é um "subtipo" do tipo inteiro
  ↳ True (1) e False (0)

• Conversão do Tipo Explícita

Conversões podem ser feitas de modo explícito usando funções construtoras

ex: `>>> int (3.4)`
`3`

## Aula 10 Biblioteca padrão Python

• Biblioteca Padrão Python

milhares de funções e classes organizadas em componentes chamados módulos

contém um conjunto de funções e/ou classes relacionadas a determinado domínio de aplicações

DATA 24 03 22

- Biblioteca Padrão Python
  - Inclui módulos para suporte a:
    - * Programação em rede
    - * Programação de aplicação Web
    - * Desenvolvimento de interface gráfica com o usuário (GUI)
    - * Programação de banco de dados
    - * Funções matemáticas
    - * Geração de números pseudoaleatórios

- módulo math
  - módulo math é uma biblioteca de constantes e funções matemáticas (ex: raiz quadrada, funções trigonométicas)
  - Importar o módulo:

```
>>> import math
```

↳ coloca todas as funções matemáticas à disposição para uso

  - Algumas funções e constantes do módulo math:

| | |
|---|---|
| sqrt (x) | $\sqrt{x}$ |
| ceil (x) | $\lceil x \rceil$ (ou seja, o menor inteiro $\geq x$) |
| floor (x) | $\lfloor x \rfloor$ (ou seja, o maior inteiro $\leq x$) |
| cos (x) | cos (x) |
| sin (x) | sen (x) |
| log (x, base) | $\log_{base}(x)$ |
| pi | 3,141592... |
| e | 2,71828... |

DATA 24 / 03 / 22

- módulo fractions

- módulo fractions torna disponível um novo tipo de número: o tipo fraction, usado para reprensentar frações e realizar aritmética racional

Importar o módulo:

```
>>> import fractions
```

```
ex:>>>a = fractions.Fraction (3,4)
>>> a
Fraction (3,4)
```

## Aula 11 Programas em Python

- Programa em Python

- Conjunto de instruções que são executadas em ordem, seguindo conceitos de algoritmos:
  - Sequenciamento
  - estruturas de condição
  - estruturas de repetição

- Implementar as instruções de um programa em um ou mais arquivos com extensão .py
  - usar editor disponível na IDE utilizada
    - IDE IDLE
    - IDE PyCharm
    - Linha de Comando

DATA 31 03 22

## Aula 12 — print(), input() e eval()

- print()

- Imprimir aquilo que é enviado no argumento (entre parênteses)
  - ↳ usuário pode visualizar algo
    ex: mensagem, resultado de uma operação

- A função pode aceitar uma quantidade qualquer de objetos de entrada, não necessariamente do mesmo tipo. Utiliza-se para isso vírgulas
  - ⊛ os valores serão impressos na mesma linha e com espaços em brancos inseridos entre eles

```
ex: >>> h = 13
    >>> print ('Agora são', h, 'horas')
    Agora são 13 horas
```

- Separador padrão: espaço em branco
  - ↳ para utilizar outro separador, usa o argumento de separação opcional sep

```
ex: >>> dia = 31
    >>> mês = 03
    >>> print (dia, mês, sep='/')
    31/03
```

  - ↳ exibir valores em linhas separadas: '\n'

```
    >>> print (dia, mês, sep='\n')
    31
    03
```

CICERO

31 03 22

- print( )
  - Imprimir um caractere no final da linha: end
    - o padrão é \n, uma quebra de linha

```
ex: >>> print (dia, mês, sep='/', end='.\n')
    >>> 31/03.
```

- input()
  - Utilizado para captura de dados digitados pelo usuário para que possam ser processados dentro do programa.
    - Lê a entrada que o usuário digitou e armazena o valor em uma variável
      - sempre retorna uma String

```
ex. >>> nome = input ('Digite seu nome')
    Digite seu nome: Mariana
    >>> nome
    'Mariana'
```

- eval( )
  - Utilizado para pedir explicitamente ao Python para avaliar o que o usuário digita como uma expressão Python.
  - eval() aceita uma String como entrada como se fosse uma expressão Python
    - pode ser utilizada junto com input() quando espera-se que o usuário digite uma expressão (um número, uma lista) quando solicitado

```
ex: >>> x = eval(input('Digite x: '))
    Digite x: 5
    >>> x == 5
```

pode usar direto o tipo int(input)/float(input)

CICERO: let (input)

31 03 22

## Aula 13 Definição de funções

- Definição de funções

- Python permite que o próprio usuário crie suas próprias funções.
- Utilizado quando precisa executar uma sequência de instruções várias vezes, em diferentes partes do programa.
  - Modularizar o programa
    - Facilita reuso e manutenção

```
def <nome da função> (<0 ou mais variáveis>):
    <instruções com identação>
    ...
    return <valor>  (opcional)
```

ex: FUNÇÃO: $f(x) = x^2 + 1$

```
def f(x)
    res = x ** 2 + 1
    return res
```

OUTRO EXEMPLO:

```
def juros (preço, juros):
    res = preco *(1+ (Juros/100))
    return res
>>> juros (10,50)
15.0
```

## Aula 14 Documentação de Programas

- Documentação

- Função da documentação:
  - que os usuários entendam o que o programa faz
  - que desenvolvidores entendam como o programa funciona

DATA 01 04 22

- Documentação

- Documentar um código em Python normalmente é realizado por meio de comentários
  - qualquer coisa que vem após o símbolo # em uma linha
  - explicar partes complexas do programa

**Exemplo:**

```
def f(x):
    res = x**2 + 1  # calcula e armazena o resultado em res
    return res      # retorna o valor de res
```

- Docstrings

- Utilizado para documentar funções para os usuários das mesma. Utiliza a função help( )
- O comentário que descreve a função é chamado de docstrings e vem logo abaixo da função, o comentário deve vir através de aspas simples

## Aula 15 Estruturas de condição de uma ou duas vias

- Estruturas de seleção

- Permite escolher um conjunto de ações
  - dependendo de uma condição ser ou não satisfeita
    - expressões lógicas ou relacionais

- Estruturas de seleção
  - Seleção de uma via (se a condição for V, executa o bloco)
  - Seleção de duas vias (V executa um bloco, F executa outro bloco)
  - Seleção de três ou mais vias (várias condições)

DATA 23 04 22

- Seleção de uma via

  - Usada para testar uma condição antes de executar uma instrução. Se a condição for avaliada como true, o código é executado, se for avaliada como false, o código é pulado.

  ↳ expressão true or false

```
if <condição>:
    <bloco de código endentado>
<instrução não endentada>
```

- Seleção de duas vias

  - Usada quando é necessário realizar uma ação quando a condição é verdadeira e outra se a condição é falsa.

```
if <condição>:
    <bloco de código endentado 1>
else:
    <bloco de código endentado 2>
<instrução não endentada>
```

## Aula 16 Estruturas de condição de três ou mais vias

- Seleção de três ou mais vias

  Adiciona mais condições para avaliação antes de executar uma instrução.

```
elif = "else if"
```

23 04 22

- Seleção de três ou mais vias

```
if <condição>:
    <bloco de código endentado 1>
elif <condição 2>:
    <bloco de código endentado 2>
    ⋮
elif <condição 'n'>:
    <bloco de código endentado 'n'>
else:
    <último bloco de código endentado>
<bloco de código não endentado>
```

## Aula 17 Estruturas de repetição - for

- Estruturas de repetição (loops)
  - Estruturas de controle de fluxo que permitem repetir uma sequência de comandos
    - número de repetições pode ser indeterminado, porém deve ser finito

- Comando 'for'

```
for <variável> in <sequence>:
    <bloco de código endentado>
<bloco de código não endentado>
```

- O comando for pode iterar uma lista e, em cada repetição ele vai assumindo valores da lista
- <sequence> deve ser: tipo string, list, range ou outro que possa ser iterado

**EXEMPLO:**

```
L = ['cão', 'gato', 'boi']
for i in L:
    print(i)
```

```
cão
gato
boi
```

28 04 22

- Função `range()`

- Pode ser utilizada junto de 'for' para percorrer uma sequência de números em determinado intervalo.

`range(n)` usada para percorrer uma sequência de inteiros de 0 até n-1

`range(início, fim)` usada para percorrer sequências que começa em um número diferente de 0 (início) e termine antes do número fim.

`range(início, fim, passo)` usada para percorrer a sequência de inteiros começando em início, usando um tamanho de passo e terminado antes do número fim.

**EXEMPLOS:**

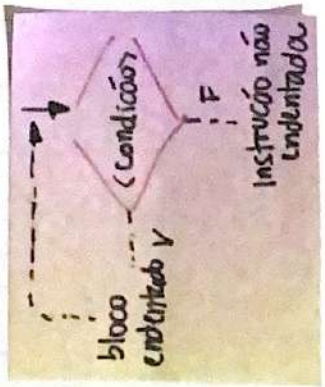
```
>>> for i in range(3):
        print(i)
0
1
2
>>> for i in range(2,5):
        print(i)
2
3
4
>>> for i in range(1,14,3)
    print(i)
1
4
7
10
13
```

## Aula 18 Estruturas de repetição - while

- Comando 'while'

- Utilizada quando queremos que determinado bloco de código seja executado enquanto determinada condição for verdadeira.

```
while <condição>
        <bloco de código endentado>
<instrução não endentada>
```




**EXEMPLO**

```
num = eval(input('Digite
    um número positivo: '))
while num < 0:
    num = eval(input(
        'Digite um número
        positivo: '))
```

28 04 22

- Loop infinito

- O laço while pode ser utilizado para criar um loop infinito, ou seja. um laço executado "para sempre". Utilizados em programas que fornecem um serviço indefinitivamente.
  Sair do loop: Ctrl + C

```
while True:
    <bloco de instruções identado>
```

## Aula 19 Outros comandos: break, continue e pass

- Comando 'break'

- Quando inserido dentro de um loop/laço (for ou while) faz com que o laço seja finalizado, passando o controle para o próximo bloco de instruções não identado.

```
for <variavel> in <sequence>:
    #codes inside for loop
    if condition:
        break
    #codes inside for loop

#codes outside for loop
```

DATA 28 04 22

- Comando 'continue'

  - Quando inserido dentro de um loop (for ou while) faz com que a nova iteração do laço seja forçada a partir daquele ponto onde o comando foi inserido.

```
for <variavel> in <sequence>:
    # codes inside for loop
    if condition:
        continue
    # codes inside for loop
# codes outside for loop
```

- Comando 'pass'

  - Os comandos def, if, else, for ou while precisam ter um bloco identado não vazio, quando o código de blocos não precisa fazer nada, tem que se colocar um código, então usa-se o 'pass'.
    - ↳ se ainda não sabe o que colocar no bloco também pode usar 'pass'

## Aula 20 Listas Multidimensionais

- Listas Multidimensionais

  - Matrizes são estruturas bidimensionais (tabelas) com m linhas por n colunas.
    - ↳ Em Python, uma matriz pode ser representada como uma lista de listas

28 04 22

- Listas Multidimensionais

Lista de listas

↳ um elemento da lista contém uma linha da matriz, que corresponde a uma lista com os elementos da coluna da matriz

```
>>> t=[[4,7,2],[5,1,9],[8,3,6]]
```

| | t[0][0] | t[0][1] | t[0][2] |
|---|---|---|---|
| t[0] | 4 | 7 | 2 |

| | t[1][0] | t[1][1] | t[1][2] |
|---|---|---|---|
| t[1] | 5 | 1 | 9 |

| | t[2][0] | t[2][1] | t[2][2] |
|---|---|---|---|
| t[2] | 8 | 3 | 6 |