

Apresentação

Python e Teste



Python e Teste

Porque usar ?

Python

- Linguagem de programação multi-
 - Uma comunidade ampla;
 - Web, ciência de dados, simulação, comunicação, e imagem
- Tarefas complexas são executadas com um único comando
 - Programação fácil (escrita e leitura)
 - Acervo grande de funções (bibliotecas por áreas)
 - Integração com outras linguagens (C, C++, Java e Javascript)
- Um passo apenas
 - Linguagem interpretada;
 - O código produzido é executado rapidamente

Um bom testador...

- Habilidades fundamentais
 - Front end: CSS, HTML, script
 - Automação: webDriver, java, javascript, python, Ruby
 - Manuseio de código: git, git-*, BASH
- Frameworks para Automação de teste
 - Robot: (para que?)
 - PyTest:
 - PyUnit:
 - Selenium:



Canais de interação

- Grupo no Telegram
- Repositório no GitHub @cavmelo/IARTES



Modulo #1

Agenda

- Instalação
- Ambiente de programação
- Primeiro programa/comentário
- Tipos de dados
- Operadores
- Variável
- Entrada e saída



Instalação

- Tem passo a passo disponível para cada plataforma.
 - Windows: <https://python.org.br/instalacao-window>
 - Linux: <https://python.org.br/instalacao-linux>
 - Mac: <https://python.org.br/instalacao-mac>
- Instale a versão 3 (python3)
 - Python3
- Na linha de comando:
 - Python --version
 - Saída esperada: Python 3.x.x
 -



Ambiente de programação

- Crie uma pasta (e.g. modulo_python)
- Instale um ambiente virtual
 - Dicas no arquivo Ambiene.md (repositório GIT)
- Instale uma IDE
 - Dica: PYCharm
 - <https://www.jetbrains.com/pycharm/download>
- Associe o ambiente virtual ao IDE
 - Depende da IDE usada
 - Garanta que os ambientes estejam associados



Primeiro programa

- Abra o seu ambiente de programação
 - IDE para quem instalou
 - Editor de tex
- O clássico "olá Python"
- Linguagem interpretada
 - Execute o seu código sendo o resultado esperado a mensagem





Tipo de dado

Tipos de dados

- Inteiros - int
- Ponto flutuante - float
- Número complexo - complex
- Booleano - bool
- Strings - str
- Lista - list
- Dicionários - dict
- Tuplas - tup
- Conjuntos - set
- Representação do tipo de valor que se tem no contexto das computações que um programa pode realizar;
- Tipos representam conceitos oriundo da matemática
- Associados a palavras especiais;



Inteiros

- Representa conceito definido na matemática;
 - Sem parte fracionária
 - Exemplos: 3, 10, -15, -100
- O inteiro é representado por **int**
- Por ser positivo e negativo
- Como é usado?
 - `idade = int(input())`
 - `Idade: int`



Ponto flutuante

- Representa conceito definido na matemática;
 - Exemplos: 3.141516, 10.1924 e 1.2550
- O ponto flutuante é representado por **float**
- Por ser positivo e negativo
- Como é usado?
 - `peso = float(input())`
 - `peso: float`



Número complexo

- Representa conceito definido na matemática
 - Exemplos: $3+3j$
 - O número complexo é representado por **complex**
 - Existe uma parte real e uma imaginária
- Como é usado?



Booleano

- Representa dois valores
 - True - verdadeiro
 - False - falso
- O booleano é representado por **bool**
- **Construção de expressões lógicas**
- Como é usado?
 - cansado:bool
 - cansado = False



Strings

- Coleção de caracteres
- Usa-se aspas simples ou dupla para caracterização da string
 - "Introdução a python"
 - 'Python é uma interessante'
- A string é representado por **str**
- Como é usado?
 - endereco = 'Rua A, 100'



Lista

- Coleção de elementos (ordenado)
 - Pode conter elementos duplicados
 - caracterização da **lista**
 - []
 - ['Python' , "interessante"]
 - A lista é representado por **list**
- Como é usado?
 - compras:list
 - compras = ["arroz", "carne", "verduras"]
 - meu_estoque: list = ["arroz", 2,5.20, "feijão", 2, 3.90"]



Conjunto

- Coleção de elementos (não ordenado)
 - Não permite elementos **duplicados**
 - caracterização da **conjunto**
 - {}
 - {'Python', 'interessante'}
 - O Conjunto é representado por **set**
- Como é usado?
 - compras:set
 - compras = {"arroz", "carne", "verduras"}
 - item_estoque:set = {"arroz", 2,5.20}



Dicionário

- Coleção de elementos (não ordenado) de pares chave-valor
- Caracterização do **dicionário**:
 - **{"chave": "valor associado"}**
 - {'Python': 'interessante'}
- O Dicionário é representado por **dict**
- Como é usado?
 - compras: dict
 - compras = {"arroz": "2kg", "carne": "3kg", "verduras": "1mc"}
 - precos: dict = {"arroz": "4.0", "frango": "10.0"}



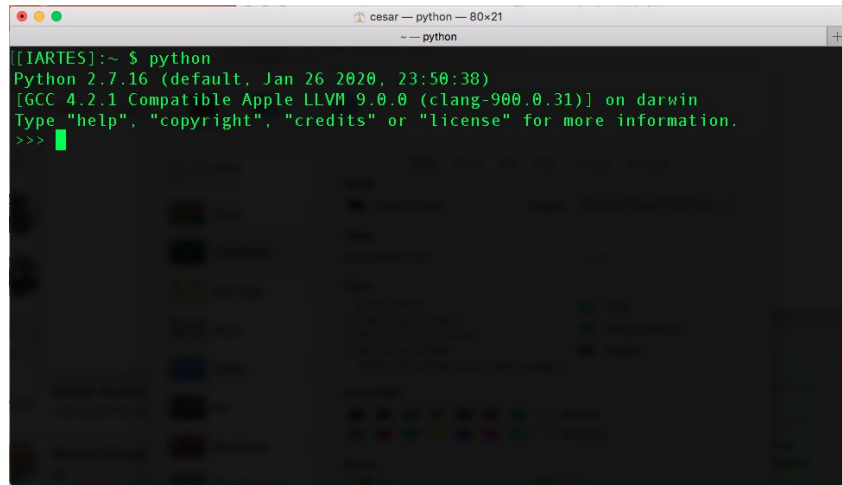
Tuplas

- Coleção de elementos (ordenado) de objetos
- Tuplas são **imutáveis**
- Caracterização da **tupla**
 - `()`
 - `('Python','interessante', 2.8)`
- A Tupla é representada por **tuple**
- Como é usado?
 - comprado: tuple
 - comprado = ("carro",0K, 2022)
 - gasolina: tuple = ("normal", 5.79)



Vamos interagir com Python shell

- Tipos de dados (type)
- Verifique o tipo inteiro
- Verifique o tipo real
- Verifique o tipo string
- Verifique o tipo dicionário
- Verifique o tipo tupla
- Verifique o tipo conjunto
- Verifique o tipo lista



```
cesar — python — 80x21
~ — python
[[IARTES]:~ $ python
Python 2.7.16 (default, Jan 26 2020, 23:50:38)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Operadores

Operadores

- Operadores são símbolos especiais que indicam que certa computação deve ser realizada.
 - Os valores envolvidos em uma operação são chamados de **operandos**
 - Exemplos:
 - $a = 10$
 - $b = 20$
 - $a + b$
- Operandos podem ser variáveis (vamos já ver) e/ou literais
- Uma sequência de operandos e operadores é chamada de **expressão**
 - $a + b - 10$



Operadores Aritméticos

Operador	Exemplo	Significado	Resultado
+(unário)	+a	unário positivo	a
+(binário)	a + b	adição	Soma de a e b
-(unário)	-a	unário negativo	Inverte o sinal do valor
-(binário)	a - b	subtração	b subtraído de a
*	a * b	multiplicação	Produto de a e b
/	a / b	divisão	Quociente quando a é dividido por b. resultado é sempre um float

Operadores Aritméticos (cont.)

Operador	Exemplo	Significado	Resultado
%	$a \% b$	módulo	Resto quando a é dividido por b
//	$a // b$	divisão inteira	Quociente quando a é dividido por b . O valor é o menor inteiro mais próximo.
**	$a ** b$	Exponenciação	a na potência b .

Vamos interagir com o Python Shell

- `a = 10`
- `b = 3`
- `+a` (unário positivo)
- `a + b`
- `-b` (unário negativo)
- `a - b`
- `a * b`
- `a / b`
- `a // b`
- `a ** b`
- `a % b`

- Verifique os tipos de dados dos resultados
 - ◆ Lembre-se: **`type(a+b)`**
- Verifique o valor resultante das operações
- Atribua um valor real a variável **`b`**, e veja se ocorre alteração no tipo de dado dos resultados



Cuidado...

Precedência	Operador	Descrição
Baixa	+, -	Soma e subtração
	*, /, //, %	Multiplicação, divisão, divisão inteira, módulo
	+x, -x	Unário (negativo e positivo)
Alta	**	Potenciação

- $a = 10$
- $b = 3$
- $a + b * 3$
- $(a + b) * 3$
- $a - b * 3$
- $(a - b) * 3$
- $a - b ** 2$
- $(a - b) ** 2$



Variável

Variável

Na programação de computadores, variáveis armazenam informações que serão referenciadas e usadas por programas.

- **Tem tipo**, exemplo: float, int, str e bool;
- **Tem nome**, formado por letras, alguns símbolos, e números;
- **Tem sensibilidade**, diferença entre Maiúscula e minúscula;
- **Tem contexto**, define a visibilidade do nome;

- Exemplos de definição:

- idade = 21
- type(idade)
- 1idade = 21
- idade1 = 21
- \$idade = 21
- idade\$ = 21
- IDADE = 21



Variáveis (cont.)

- É possível atribuir valores para múltiplas variáveis em uma linha;
 - É possível atribuir um valor para múltiplas variáveis;
 - Use o ***print()*** para imprimir os valores armazenados em uma variável
- `x, y, z = "laranja", "banana", "tucumã"`
 - `x = y = z = "tucumã"`
 - `print(x, y, z)`
 - `print(x + y + z)`



Entrada e Saída

Entrada e Saída

- Permite receber informações para realizar uma computação.
 - Entrada padrão é o teclado
 - Usa o comando `input()`;
 - **`cidade = input()`**
- A informação lida precisa ser tratada
 - String por padrão
 - Outros tipos precisam ser convertidos
 - `idade = int(input())`
 - `preco = float(input())`
- Escreva o script que faça a leitura de um nome próprio e depois imprima o valor lido.
- Tente fazer a leitura de dois nomes próprios usando duas variáveis diferentes na mesma linha.



Entrada e Saída

- Dados podem ser lidos de um arquivo
- O arquivo precisa estar disponível para a leitura
 - `f = open("meuarquivo.txt", "r");`
 - O primeiro parâmetro é o nome_do_arquivo;
 - O segundo parâmetro é o modo de leitura:
 - `"r"` - Read (leitura)
 - `"a"` - Append (escrita final)
 - `"w"` - Write (abre para escrita)
 - `"x"` - Create (cria o arquivo)
- O nome do arquivo pode conter o diretório (pasta) onde o arquivo está localizado;
- `f.read()` e `f.readline()`



Entrada e Saída

Permite que resultados de computações sejam enviados para a saída padrão (tela).

- O comando para realizar a saída de dados na tela é o **print()**
- Aceita qualquer tipo de dados
 - nome='Maria Júlia'
 - print(nome)
- Aceita imprimir resultados de expressões
 - nome='Maria Júlia'
 - print(nome + " da silva")



Entrada e Saída

Saída também pode ser feita para arquivo. Nesse caso é preciso ter um arquivo aberto para escrita.

- Pode ser feito usando o comando ***open()***
 - `f = open("meuarquivo.txt", "w")`
 - `f.write("olá mundo")`
- A escrita ocorre da esquerda para direita, de cima para baixo.
- É possível deslocar o ponteiro de escrita para qualquer parte:
 - `f.seek(2)` → escrita a partir do byte 2



Atividade

- Acesse o arquivo docs/exerc-m01.md no github, repositório @cavmelo/IARTES.
- Responda as questões produzindo um arquivo .md, e o publique no repositório;
- Identifique o seu trabalho usando a sua matrícula como nome do arquivo.



Modulo #2

Condicional

Projeto Sobre cobertura de código

Processando a saída do coverage

Em um dicionário (*dict*) Python foi armazenado o resultado dos testes realizados em um software. Processe o dicionário e indique quais módulos tem uma cobertura de código **menor** que um certo percentual.

Entradas:

1. A estatística de cobertura em um arquivo;
2. O limite do percentual de cobertura;

Saída:

- Nome dos módulos com cobertura menor que o percentual desejado. Informe o nome do módulo e o percentual de cobertura (por exemplo, 80%). É possível que tenham mais que um módulo.

Agenda

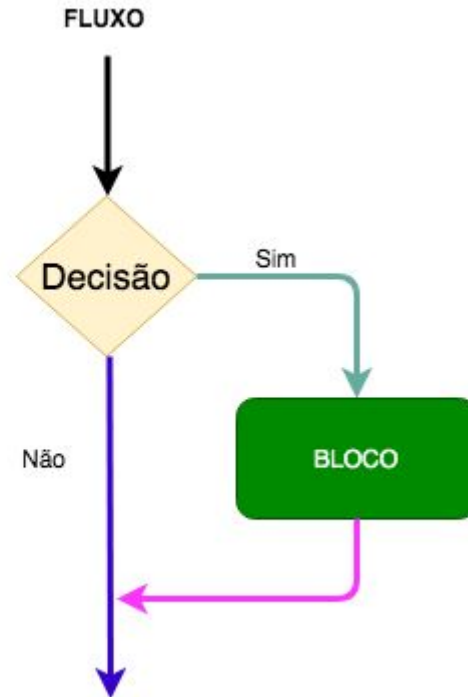
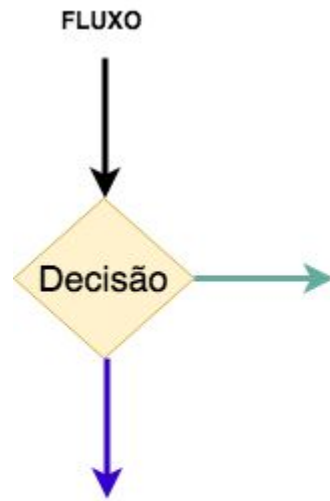
- O que computar?
 - If
 - If else
 - If elif else
- Repetindo uma computação
 - For
 - While
 - do
- Reaproveitando código próprio
 - Funções
 - Contexto local e contexto global
- Módulos
 - Como usar código desenvolvido por terceiros



Estrutura de decisão

Permite alterar a direção do fluxo
de execução

Decisão Simples



Situação para usar uma decisão simples

1. Estacionou o carro na calçada então tem multa;
2. A velocidade atual é maior que o limite estabelecido para a via então tem multa;
3. Salto completado é maior que o recorde mundial então temos novo recordista;
4. Chegou a hora de tomar água então ingerir um 300ml e ativar novamente o contador de tempo para a próxima ingestão;

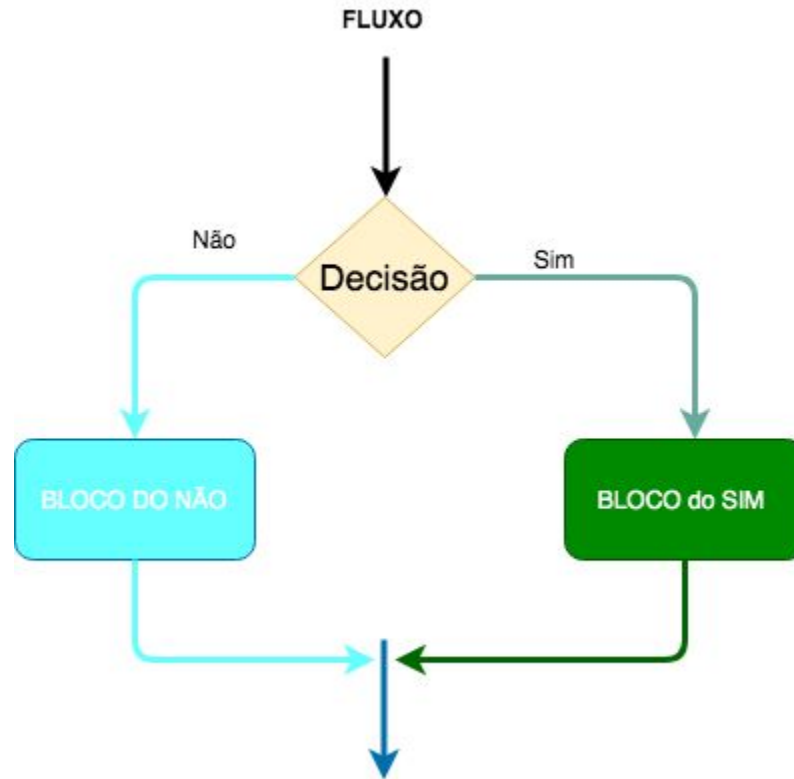
Em Python, como construir uma decisão simples?

Sintaxe :

if <expressão lógica>:

<Bloco de Comandos>

Decisão Composta



Situação para usar uma decisão Composta

1. Atingiu a maioria? Sim e Não
2. As metas de cobertura de código foram atingidas? Sim e Não
- 3.

Em Python, como construir uma decisão simples?

Sintaxe :

if <expressão lógica>:

<BLOCO SIM>

else:

< BLOCO NÃO>

Condicionais Encadeada

+Blocos

Expressar naturalmente múltiplas
escolhas que são excludentes

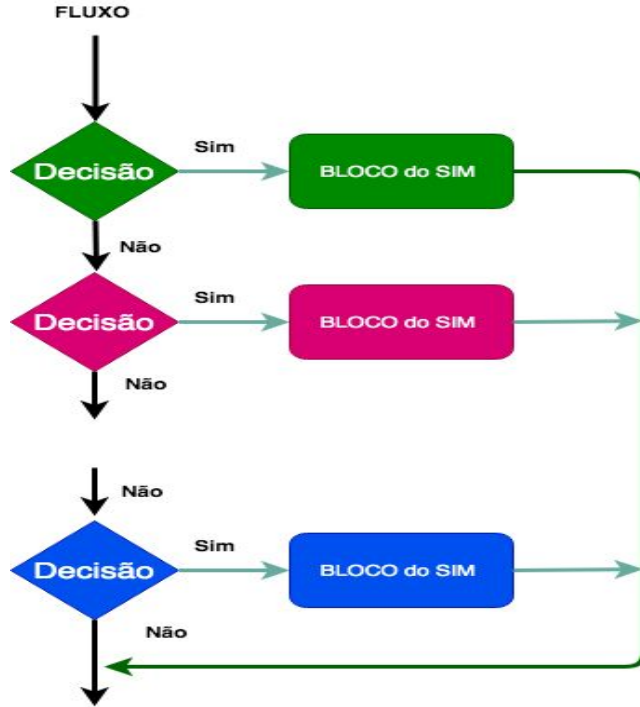
Contexto

Estruturas de decisão ENCADEADAS.

1. Opções de serviços
2. Agrupamento de objetos/pessoas;
3. Áreas do conhecimento humano;
4. Tipos de modal de transporte
5. Formas de efetuar um pagamento



Sintaxe geral



if <condição 1>:

Bloco 1

elif <condição 2>:

Bloco 2

elif <condição N>:

Bloco N

else: // Nenhuma Das Anteriores

Bloco NDA

Como expressar as condições

Operadores relacionais:

- Maior (>); Menor(<); Igual(==); diferente(!=); Maior igual (>=); Menor igual(<=)

Exemplo:

- Qual opção será usada na máquina de cobrança?

```
if opcao == 1: # crédito  
  
elif opcao == 2: # débito  
  
else: # erro na digitação
```



Como expressar as condições

Operadores lógicos:

- E (AND); OU(OR); negação(!)

Exemplo:

- Dias de vacinação para as diferentes faixas etárias?

	OP01	OP02	Resultado
AND	F	V/F	F
	F/V	F	F
	V	V	V

	OP01	OP02	Resultado
OR	F/V	V	V
	V	F/V	V
	F	F	F

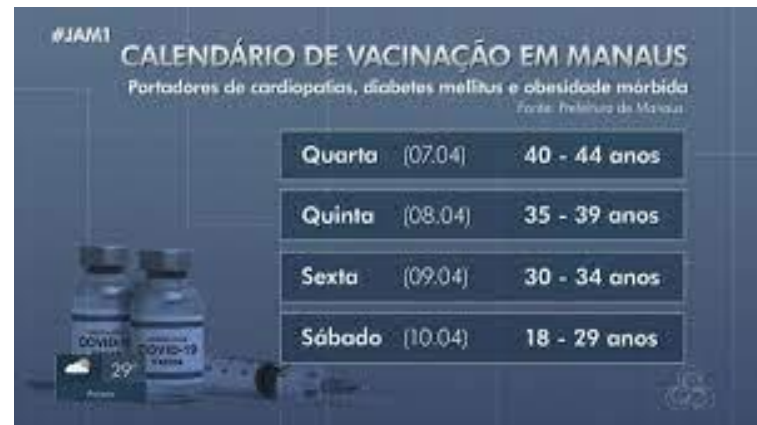
Como expressar as condições

Operadores lógicos:

- E (AND); OU(OR); negação(!)

Exemplo:

- Dias de vacinação para as diferentes faixas etárias?



#JAM1

CALENDÁRIO DE VACINAÇÃO EM MANAUS
Portadores de cardiopatias, diabetes mellitus e obesidade mórbida
Fonte: Prefeitura de Manaus

Quarta	(07.04)	40 - 44 anos
Quinta	(08.04)	35 - 39 anos
Sexta	(09.04)	30 - 34 anos
Sábado	(10.04)	18 - 29 anos

The image shows a promotional graphic for COVID-19 vaccination in Manaus. It features a calendar table with dates from April 7th to 10th, each assigned to a specific age group. In the background, there are images of COVID-19 vaccine vials and a syringe.

if idade >= 40 AND idade <=44: #Quarta

elif idade >= 35 AND idade <=39: #Quinta

elif idade >=30 AND idade <=34: #Sexta

elif idade >=18 AND idade <= 29: #Sábado

Estrutura de Repetição

Permite que um bloco de comandos seja executado repetidas vezes

Repetição

- Garante que um bloco de comando seja executado repetidas vezes.
- Torna o código capaz de resolver efetivamente problemas de tamanho diversos.
 - Tamanho é definido por uma especificidade da entrada
 - Tamanho é um parâmetro de entrada do programa



Repetição: Condiciona a execução do bloco

Enquanto - While: primeiro avalia a condição

while (<condição>):

BLOCO

```
i = 1
```

```
while (i<10):
```

```
    print("%d\n" % i*i)
```

```
    i = i + 1
```

Exemplos de problemas que precisam de estruturas de repetição.

1. Descobrir o maior número em uma sequência de valores positivos (maior que zero).
2. Contar quanto número pares existem em uma sequência de valores positivos;
3. Descobrir qual é a maior diferença entre valores consecutivos em uma sequência de números positivos. Considere a diferença em valor absoluto, ou seja,

$$10 - 1 = 9$$

$$1 - 10 = 9$$

Codificação

Descobrir o maior número em uma sequência de valores positivos (maior que zero).

- Um do modelo mental #1: Hipótese e validação para cada nova instância.
- Bloco construído em cima do caso geral;
- Tratamento de exceções ocorre com o refinamento do código

Passo 01: Ler o primeiro valor, e o define com o maior (hipótese)

Passo 02: Ler o próximo valor;

Passo 03: Validar a hipótese, ajustar se for o caso

Passo 04: Ler o próximo valor;

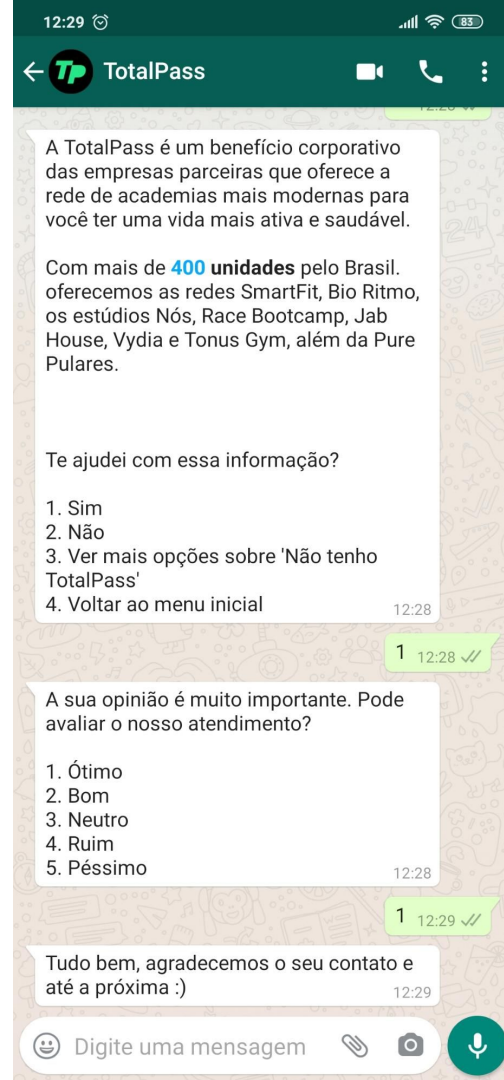
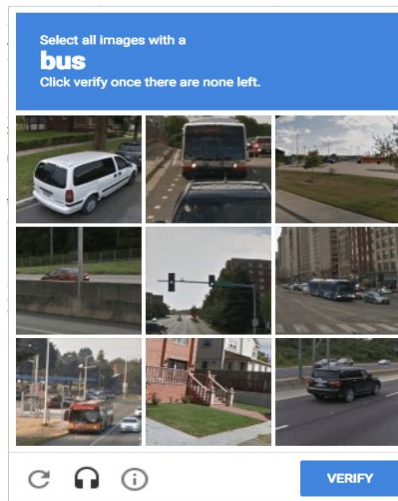
Passo 05: Validar a hipótese, ajustar se for o caso;

Passo 06: Ler o próximo valor;

Passo 07: Validar a hipótese, ajustar se for o caso;

Validando uma entrada de dados

- Passo 1: Apresentar as opções
- Passo 2: Ler as entradas de dados
- Passo 3: Validar a entradas
- Passo 4: Repetir Passo 1, caso a entrada seja inválida.



Repetição: Condiciona a execução do bloco

Contagem - for: primeiro avalia a condição

for <inicializacao> **in** <intervalo>:

BLOCO

```
for i in range(1,10):
```

```
    print("%d\n" % (i*i))
```

Exemplos de problemas que precisam de estruturas de repetição.

1. Descobrir o maior número em uma sequência com 100 valores.
2. Contar quanto número pares existem em uma sequência com 350 valores;
3. Um sensor registra a temperatura de uma sala a cada 2 segundos. Determine a maior temperatura registrada na sala ao longo de 24 horas. O registro começa sempre 0 (zero) hora.

Codificação

Descobrir o maior número em uma sequência com **100** valores.

- Um do modelo mental #1: Hipótese e validação para cada nova instância.
- Bloco construído em cima do caso geral;
- Tratamento de exceções ocorre com o refinamento do código

Passo 01: Ler o primeiro valor, e o define com o maior (hipótese)

Passo 02: Ler o próximo valor;

Passo 03: Validar a hipótese, ajustar se for o caso

Passo 04: Ler o próximo valor;

Passo 05: Validar a hipótese, ajustar se for o caso;

Passo 06: Ler o próximo valor;

Passo 07: Validar a hipótese, ajustar se for o caso;

Funções

Definição

Trecho de código que realiza uma computação específica.

- Cria níveis maiores de abstração;
- Evita que o código se torne repetitivo;
- Cada função tem uma assinatura única
 - Seu retorno;
 - Seu nome
 - Seus parâmetros;
- Parâmetros são valores demandados para a execução do trecho de código na função.

- Sintaxe Geral:

```
def <nome>(<parametros>):  
    <Comandos>
```

- Exemplos:

```
def valida_cpf(nro_cpf):  
    <Comandos>  
    return <resultado>
```

```
def atualizar(lista, item):  
    <Comando>
```

Chamada e retorno

Chamada de uma função é feita pela referência ao seu nome:

- `nome = input()`
- `print("Seu nome: %s" % nome)`

Uso de argumentos não obrigatórios

- Computação pode ser feita com entradas padrão;
- A ordem dos parâmetros na assinatura pode ser subvertida

```
def conversor_temp(graus,direcao='celsius'):
```

```
    if (direcao == 'celsius'):
```

```
        return (graus-32)*5/9
```

```
    else:
```

```
        return (graus*9/5)+32
```

```
conversor_temp(graus=32,direcao='fahrenheit')
```

Chamada e retorno

O resultado da computação realizada.

- nome = input()
- idade = int(input)

Pode ter k retornos

- É preciso casar corretamente retornos com seus destinos
- Ordem de atribuição é definida pela ordem de listagem;

- Uma função que produz a abreviação de nome próprios, retornando o resultado em três partes: Primeiro nome, nome do meio (abreviado), último nome;
- Uma função que valida uma data de nascimento. Retorna **código de validação, seguido de uma mensagem de descrição.**

Orientação a objeto


POO - Definição

É um paradigma de programação que se baseia na organização e estruturação de código em torno de "objetos", que são instâncias de classes

By chatGPT



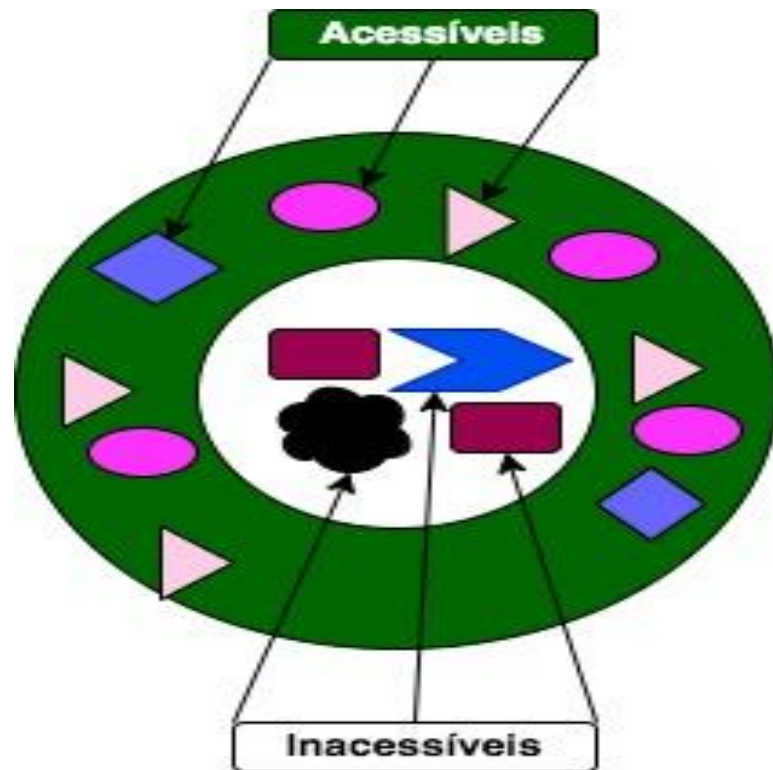
P00 - 4 princípios

- Encapsulamento
 - Ocultar detalhes da representação do conhecimento
 - Evitar acoplamento entre lógica e representação dos dados
 - Abstração
 - Capturar detalhes essenciais de conceitos do mundo real
 - Complexidade abstraída por cada representação
 - Herança
 - Definir novas classes a partir de classes existentes
 - Polimorfismo:
 - Permitir que instância da hierarquia de classes respondam de forma própria a evocação do mesmo método.
- 

Encapsulamento

Ocultar detalhes da representação do conhecimento

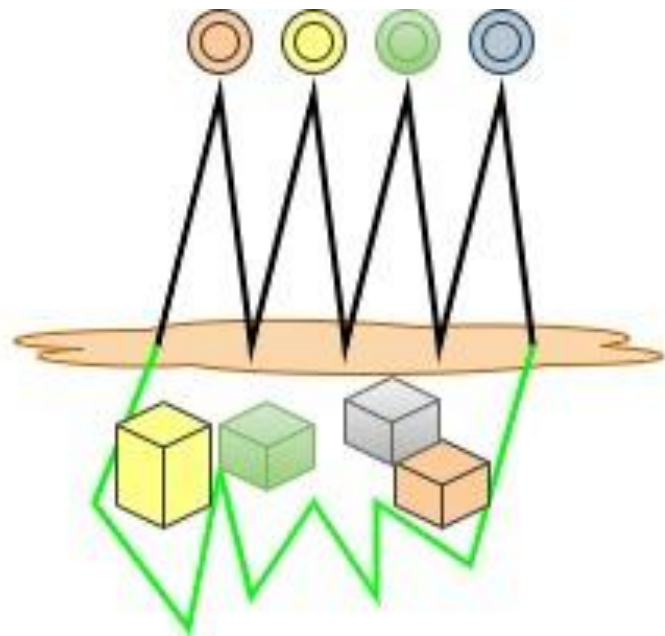
- Existem dois tipos:
 - Dados
 - Operações
- Comunicação via interface:
 - Contexto que estabelece



Abstração

Capturar detalhes essenciais de conceitos do mundo real

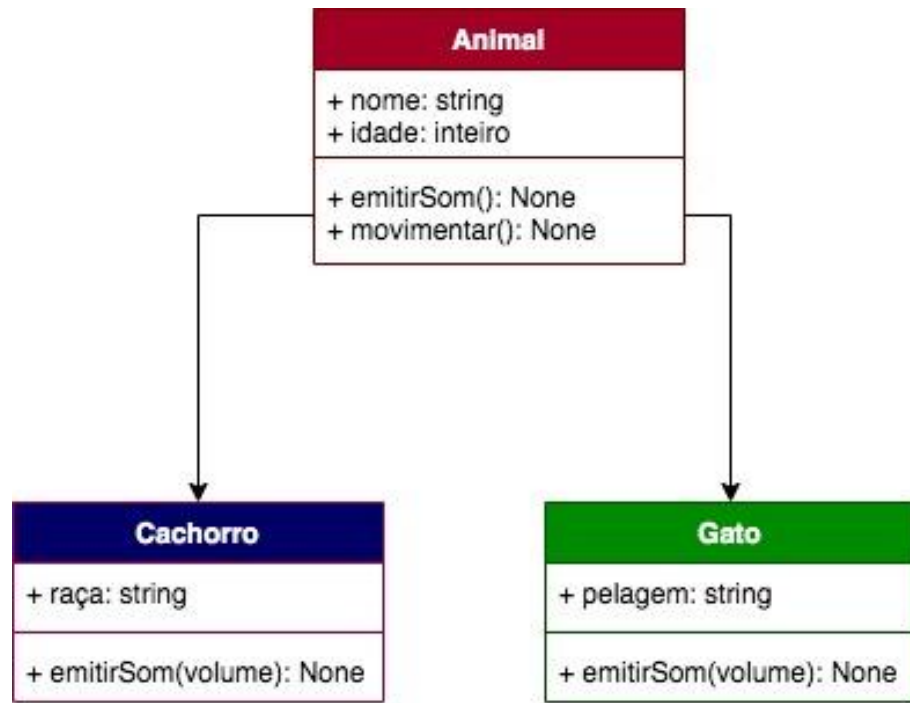
- Mantém oculto os detalhes;
- Expõe o que é essencial (métodos e dados) para o contexto do programa;



Herança

Definir novas classes a partir de classes existentes

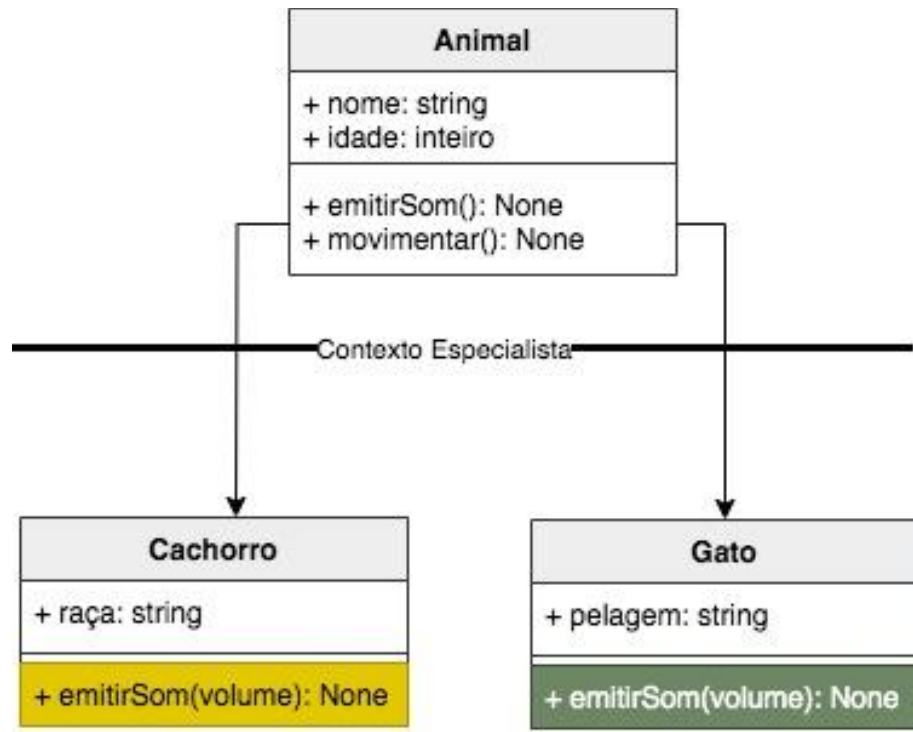
- Promove a reutilização de código
- Facilita descrição de hierarquias



Polimorfismo

Permitir que instância da hierarquia de classes respondam de forma própria a evocação do mesmo método.

- Cria um vocabulário comum permitindo descrições com alto nível de abstração
- Codificação flexível e extensível



Representação em Python

```
class nome_da_class(<ClasseBase>):  
    '''Variáveis de classe'''  
    # lista de variáveis da classe  
    #  
    '''Método construtor'''  
    def __init__(self, <lista de atributos>):  
        <inicialização dos atributos>  
  
    '''métodos acessores'''  
    def nome_do_metodo(self, <lista de parametros>):  
        <Corpo do Método>  
        # acesso das variáveis de instância  
        # e das variáveis de classe  
        #  
  
    '''métodos mutadores'''  
    def nome_do_metodo(self, <lista de parametros>):  
        <Corpo do Método>  
        # modifica variáveis de instância e  
        # classe por meio de computação
```


Encapsulamento em Python

- Dois níveis:
 - Nomes privados
 - Nomes protegidos
- Nomes privados
 - Formados com uma marca (`__`) no início
 - Acesso restrito a classe que o definiu
 - Erro
- Nomes protegidos
 - Formados com uma marca (`_`)



Tarefa

1. Defina uma classe base, por exemplo:
 - a. Animal
 - b. Veículo
 - c. Sensor
2. Use a classe escolhida para definir uma hierarquia.
 - a. Pelo menos mais um nível
 - b. Pelo menos dois métodos especializados;
3. Escreva um código que faça uso da classe definidas
 - a. Implemente e apresente um trecho de código que faça uso do polimorfismo
 - b. Implemente e apresente um trecho de código que faça uso da herança;
 - c. Implemente e apresente um trecho de código que faça uso do encapsulamento;



TDD - Desenvolvimento Dirigido por Teste

Agenda

1. Conceitos de Orientação a objeto
2. Cobertura de código
3. Pytest

