Tecnólogo em Análise e Desenvolvimento de Sistemas

Desenvolvimento Web I

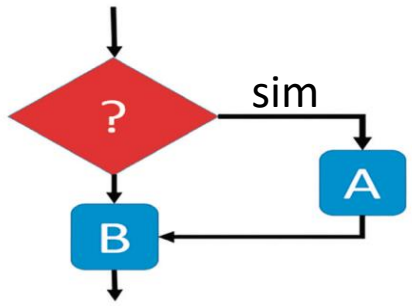
UNIVERSIDADE CANDIDO MENDES

Prof. Ricardo Tavares

ricardo.tavares@ucam-campos.br

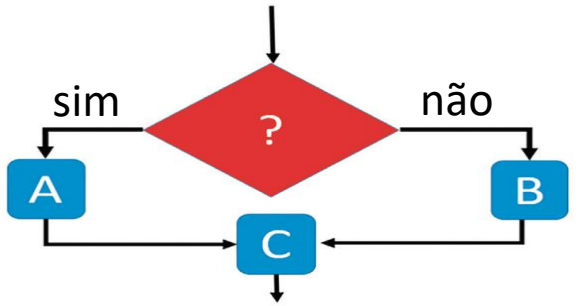
Comando if, elif, else

- Estrutura condicional
  - Executa análise sobre alguma condição:
    - Se a condição for atendida: executa;




```
if nota>=6.0:
    print('Aluno aprovado')
```

```
if condição lógica:
  #Bloco A
  Bloco de comandos executados caso a condição
  lógica seja verdadeira
#Bloco B
Comandos executados após o teste
```



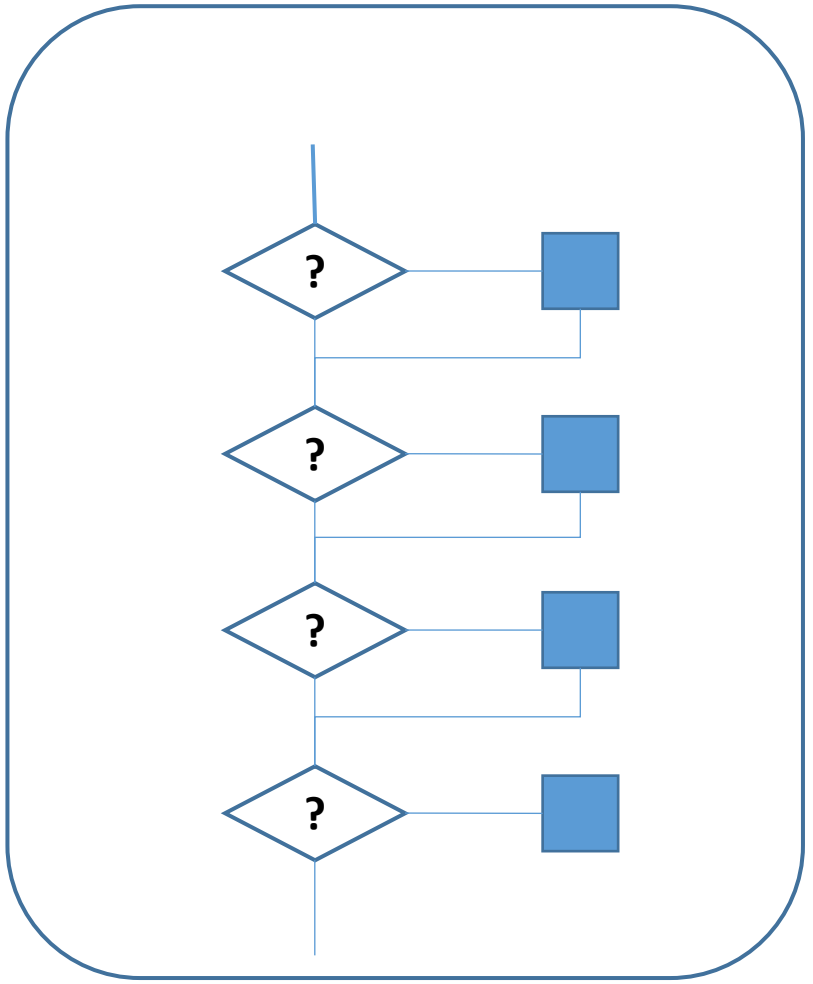

```
if nota>=6.0:
    print('Aluno aprovado')
else:
    print('Aluno reprovado')
```

```
if condição lógica:
  #Bloco A
  Bloco de comandos executados caso a condição
  lógica seja VERDADEIRA
else:
  #Bloco B
  Bloco de comandos executados caso a condição
  lógica seja FALSA
#Bloco C
Comandos executados após o teste.
```

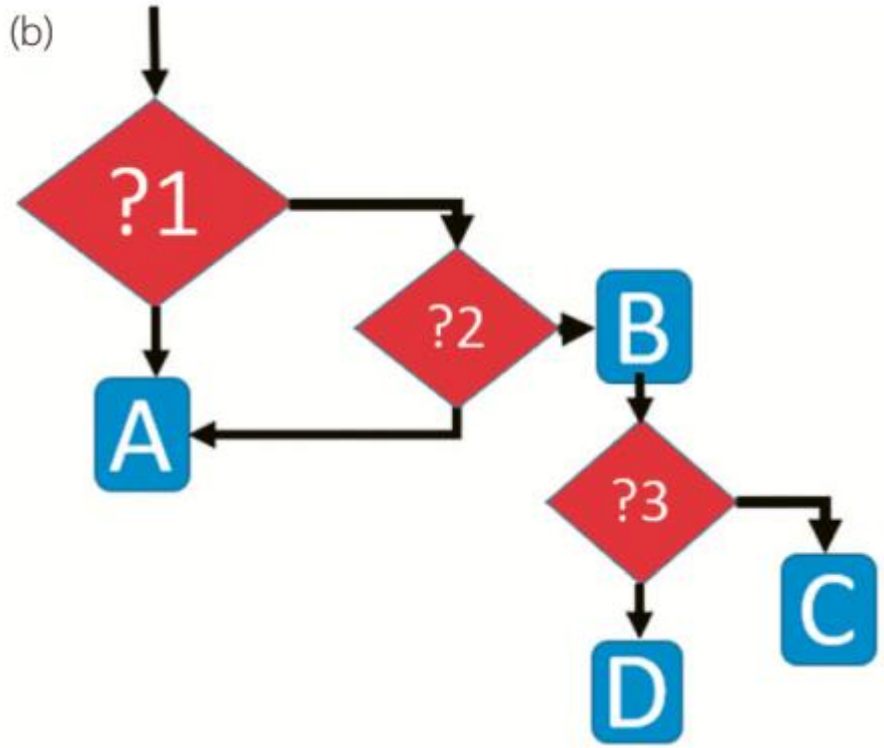
Comando if, elif, else

```
if condição lógica1:
  #Bloco A
  Bloco de comandos executados caso a condição
  lógica1 seja VERDADEIRA
elif condição lógica2:
  #Bloco B
  Bloco de comandos executados caso a condição
  lógica1 seja FALSA e a condição lógica2 seja
  VERDADEIRA
elif condição lógica3:
  #Bloco C
  Bloco de comandos executados caso a condição ló-
  gica1 seja FALSA, a condição lógica2 seja FALSA
  e a condição lógica3 seja VERDADEIRA
  ...
else:
  #Bloco N
  Bloco de comandos executados caso NENHUMA DAS
  condições lógicas sejam VERDADEIRAS

#Bloco X
Comandos executados após o teste.
```



Comando if, elif, else




```
# O Brasileirão está acirrado
# Vasco e Flamengo chegam juntos na liderança ao final
do campeonato
# Sabendo que o critério de desempate:
#     1) Número de pontos;
#     2) Saldo de gols;
# Escreva um código que receba o número de pontos e o
saldo de gols e determine o campeão

fla_pontos, fla_saldo = 70, 15
vas_pontos, vas_saldo = 70, 16

if vas_pontos > fla_pontos:
    print('Vasco campeão')
elif vas_pontos < fla_pontos:
    print('Flamengo campeão')
else:
    if vas_saldo > fla_saldo:
        print('Vasco campeão')
    elif vas_saldo < fla_saldo:
        print('Flamengo campeão')
    else:
        print('Os times terminaram empatados')
```

Comando if, elif, else

- Estrutura condicional:

Erro de indentação!

```
In [38]: nota = 4.9
if nota>=9.0:
    print('Aluno tirou nota: A')
elif (nota >= 8.0) and (nota < 9.0):
    print('Aluno tirou nota: B')
elif (nota >= 7.0) and (nota < 8.0):
print('Aluno tirou nota: C')
elif (nota >= 6.0) and (nota < 7.0):
    print('Aluno tirou nota: D')
elif (nota >= 5.0) and (nota < 6.0):
    print('Aluno tirou nota: E')
else:
    print('Aluno tirou nota: F')
```

```
  File "<ipython-input-38-199068fabd8a>", line 7
    print('Aluno tirou nota: C')
        ^
IndentationError: expected an indented block
```

Importante: para executar a ação dentro do bloco if, elif, else, deve-se criar uma indentação;

Estruturas de repetição

- Comando for:

```
for <condição>:
    <sequência de comandos>
```

```
lista = [10,11,12,13,14,15]
for item in lista:
    print(item)
```

```
10
11
12
13
14
15
```

```
cesta_de_frutas = ['banana','maça','pera','uva']
for fruta in cesta_de_frutas:
    print(fruta)
```

```
banana
maça
pera
uva
```

Estruturas de repetição

- Comando while:

```
while <condição>:
    <sequência de comandos>
```

```
lista = [5,6,7,8]
tamanho_lista = len(lista)
contador = 0
while (contador < tamanho_lista):
    print(lista[contador])
    contador += 1
```

```
5
6
7
8
```

```
cesta_de_frutas = ['banana','maça','pera','uva']
qtde_itens_cesta = len(cesta_de_frutas)
contador = 0
while (contador < qtde_itens_cesta):
    print(cesta_de_frutas[contador])
    contador += 1
```

```
banana
maça
pera
uva
```

Estruturas de repetição

- Comando break e continue:
  - O comando break, quando utilizado em uma estrutura de repetição, encerra o loop;
  - O comando continue provoca uma interrupção no loop, mas sem sair do mesmo. Ele somente avança para a próxima interação da repetição;

```
contador = 1
while contador < 10:
    contador += 1
    if (contador%2 == 1):
        continue
    print(contador)
```

```
2
4
6
8
10
```

```
contador = 1
while contador < 20:
    contador += 1
    if (contador == 15):
        break
    if (contador%2 == 1):
        continue
    print(contador, end = ' ')
```

```
2 4 6 8 10 12 14
```

Estruturas de repetição

- Estruturas aninhadas:

```
for numero in range(0,11):
    fatorial = 1
    for valor in range(1,numero+1):
        fatorial *= valor
    print('Fatorial de {}: {}! = {}'.format(numero, numero, fatorial))
```

```
Fatorial de 0: 0! = 1
Fatorial de 1: 1! = 1
Fatorial de 2: 2! = 2
Fatorial de 3: 3! = 6
Fatorial de 4: 4! = 24
Fatorial de 5: 5! = 120
Fatorial de 6: 6! = 720
Fatorial de 7: 7! = 5040
Fatorial de 8: 8! = 40320
Fatorial de 9: 9! = 362880
Fatorial de 10: 10! = 3628800
```

Listas e Tuplas

- Tuplas:
  - É uma uma coleção de dados heterogêneos (permite que seus elementos sejam de tipos diferentes).
  - Função len() -> tamanho da tupla
  - Indexação: começando do 0 à esquerda, ou de -1 à direita.
  - Slicing: x[início : fim]
  - Concatenação e replicação:
    - x *2 ; x + (5,4)
  - Imutabilidade : uma vez criada, uma tupla não pode ser alterada;

```
x = 1, 2, 3
print(x)
print(type(x))
```

```
(1, 2, 3)
<class 'tuple'>
```

```
x = (1, 2, 3)
print(x)
print(type(x))
```

```
(1, 2, 3)
<class 'tuple'>
```

```
a , b , c = x
print(a, b, c)
```

```
1 2 3
```

```
x = (1, 2, 3)
print(len(x))
print(x[0])
print(x[-1])
print(x[0:2])
```

```
3
1
3
(1, 2)
```

```
x = (1, 2, 3)
print(x)
y = x * 2
print(y)
z = x + (5, 4)
print(z)
```

```
(1, 2, 3)
(1, 2, 3, 1, 2, 3)
(1, 2, 3, 5, 4)
```

Listas e Tuplas

- Listas:
  - Também é uma coleção de dados heterogêneos (permite que seus elementos sejam de tipos diferentes)
  - Mutável: uma vez criada, uma lista pode ser modificada;
  - Função len() -> tamanho da lista
  - Indexação: começando do 0 à esquerda, ou de -1 à direita.
  - Slicing: x[início : fim]
  - Concatenação e replicação:
    - x *2 ; x + [5,4]
  - Utilize o in para saber se algum elemento pertence a lista

```
x = [1, 2, 3]
print(len(x))
print(x[0])
print(x[-1])
print(x[0:2])
```

```
3
1
3
[1, 2]
```

```
x = [1, 2, 3]
print(x)
y = x * 2
print(y)
z = x + [5, 4]
print(z)
```

```
[1, 2, 3]
[1, 2, 3, 1, 2, 3]
[1, 2, 3, 5, 4]
```

```
listaA = [1, 5.0, 'Olá', [1, 'hello']]
for item in listaA:
    print(item)
listaA[2] = "Bem-vindo"
for item in listaA:
    print(item)
```

```
1
5.0
Olá
[1, 'hello']
1
5.0
Bem-vindo
[1, 'hello']
```

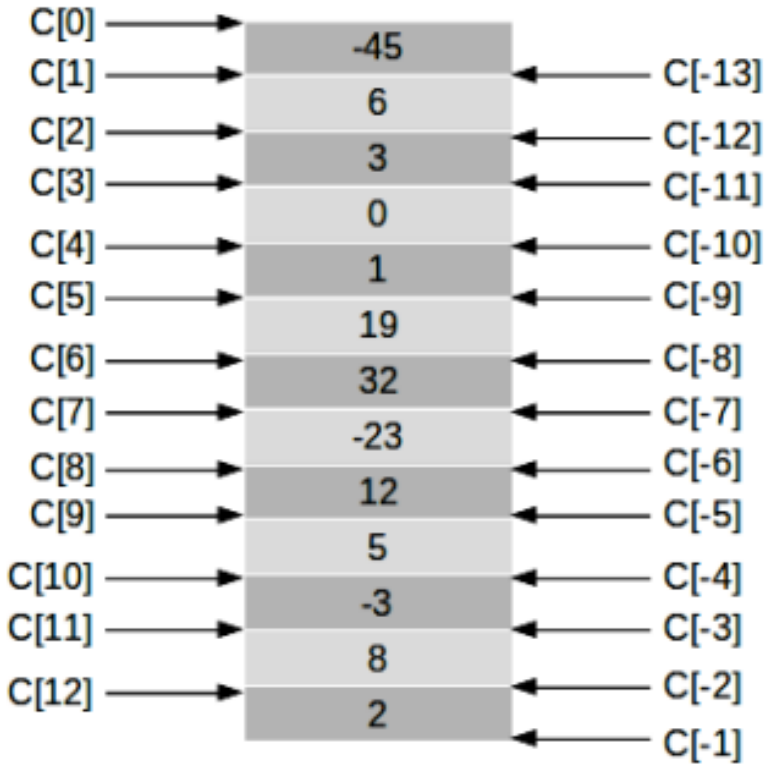
```
cesta_frutas = ['banana', 'maça', 'laranja']
print('banana' in cesta_frutas)
```

Listas e Tuplas

- Listas:
  - append() -> adiciona um item na lista na última posição;
  - pop() -> remove um item da lista, de acordo com sua posição, e o retorna;
  - remove() -> remove um item da lista , de acordo com sua posição;

```
: listaA = [1, 5.0, 'Olá', [1, 'hello']]
listaA.append(None)
print(listaA)
print(listaA.pop(1))
print(listaA)
print(listaA.remove(1))
print(listaA)
```

```
[1, 5.0, 'Olá', [1, 'hello'], None]
5.0
[1, 'Olá', [1, 'hello'], None]
None
['Olá', [1, 'hello'], None]
```



Funções

- Função é uma sequência de instruções que computa um ou mais resultados que chamamos de parâmetros;
- Os parâmetros das funções são locais: um parâmetro só existe dentro de sua função.
- Caso queira tornar um parâmetro global utilize essa palavra reservada;
- Utilize return para retornar valores através da função;

```
def "nome da função"("parâmetros"):
    '''
    docstring contendo comentários sobre a
função.
    Embora opcionais são fortemente
recomendados. Os comentários
    devem descrever o papel dos parâmetros e
o que a função faz.
    '''
    # corpo da função
    |
    | bloco de comandos
    |
```

Funções

- Apresentação de exemplos: jupyter notebook
- Exercício:
  - Escreva uma função que recebe dois inteiros, m e n, como parâmetros e retorna a combinação m!/((m-n)!n!).

Funções

- Passando um número arbitrário de parâmetros (*args):
  - Quando queremos passar vários parâmetros sem necessariamente declará-los

```
def foo(arg, *args):
    print('primeiro argumento normal: {}'
          .format(arg))
    for arg in args:
        print('*Args: {}'.format(arg))
foo('Mostrando ', 'como ', 'funciona ', 'o *args')
```

```
primeiro argumento normal: Mostrando
*Args: como
*Args: funciona
*Args: o *args
```

```
def foo(arg, *args):
    print('primeiro argumento normal: {}'
          .format(arg))
    for arg in args:
        print('*Args: {}'.format(arg))
foo([1,2,3], *['arg1', 'arg2', 'arg3'])
```

```
primeiro argumento normal: [1, 2, 3]
*Args: arg1
*Args: arg2
*Args: arg3
```

Funções

- Passando um número arbitrário de chaves (*kwargs):
  - Permite a passagem de um tamanho variável de argumentos com suas palavras-chaves (Keys)

```
def foo(arg, **kwargs):
    print('primeiro argumento normal: {}'
          .format(arg))
    for key, value in kwargs.items():
        print('chave: {} // valor: {}'
              .format(key, value))
dicionario = {'nome': 'Anakin Skywalker', 'idade': 42}
foo('Bem-vindos ao kwargs',**dicionario)
```

```
primeiro argumento normal: Bem-vindos ao kwargs
chave: nome // valor: Anakin Skywalker
chave: idade // valor: 42
```

Funções

- Passando um valor padrão para o caso de ausência de atribuição:

```
def foo(txt1, txt2='bom dia'):
    print(txt1, txt2)


foo('Bem-vindos alunos - ')
```

```
Bem-vindos alunos -  bom dia
```

```
def foo(txt1, txt2='bom dia'):
    print(txt1, txt2)


foo('Bem-vindos alunos - ', 'boa noite')
```

```
Bem-vindos alunos -  boa noite
```

Funções

- Exercícios:
  - Crie uma função que receba 3 argumentos (notas) e retorne a média de um aluno;
  - Faça a mesma coisa usando *args;
  - Crie uma função que receba 4 argumentos (notas e pesos) e retorne a média ponderada de um aluno;
  - Agora utilize **kwargs passando da seguinte forma:
    - {‘notas’: [10.0, 8.0], ‘pesos’: [3, 5]}

Classes e objetos

Classes e objetos - exemplo - Conta bancária

- Em nosso primeiro exemplo, teremos como informação de uma conta apenas *saldo* e *numero da conta*

```
1 class Conta:
2     def __init__(self, numero, saldo=0):
3         self.saldo =saldo
4         self.numero = numero
```

Classes e objetos

Classes e objetos - exemplo - Conta bancária

```
1 class Conta:
2     def __init__(self, numero, saldo=0):
3         self.saldo =saldo
4         self.numero = numero
```

- Usamos a palavra **class** para indicar a definição de uma classe.
- A classe **Conta** tem dois **atributos** (*saldo* e *numero*)
- _init_ é um método especial, denominado **construtor** por ser chamado sempre que um objeto da classe é criado (instanciado).
- o método _init_ recebe um parâmetro chamado *self*, que indica o objeto que está sendo criado.

Classes e objetos

Classes e objetos - exemplo - Conta bancária

```
1 class Conta:
2     def __init__(self, numero, saldo=0):
3         self.saldo =saldo
4         self.numero = numero
```

- *self.saldo* indica o atributo *saldo* da conta que está sendo criada. Se não colocássemos *self.* na frente, saldo seria uma variável local e teria seu valor jogado fora quando o método acabasse de ser executado.

- no interpretador, podemos criar um objeto **conta** e atribuí-lo a uma variável. Para isso, usamos o nome da classe e os parâmetros indicados no método construtor __init__.

```
1 In [1]:  c1=Conta(1234, 100)
```

Classes e objetos

Classes e objetos - exemplo - Conta bancária

- Adicionaremos a nosso exemplo o comportamento da conta ao reagir às operações de *saque* e *deposito*, além da operação *resumo* para ver o saldo.

```
1  class Conta:
2      def __init__(self, numero, saldo=0):
3          self.saldo =saldo
4          self.numero = numero
5
6      def resumo(self):
7          print("CC numero: %s Saldo: %10.2f" % (self.numero, self.saldo))
8
9      def saque(self, valor):
10         if self.saldo>=valor:
11             self.saldo -= valor
12
13     def deposito(self, valor):
14         self.saldo += valor
```

- A classe **Conta** tem agora dois **atributos** (*saldo* e *numero*) e quatro **métodos** (__init__, *resumo*, *saque* e *deposito*)

Classes e objetos

Classes e objetos - exemplo - Conta bancária

- o primeiro parâmetro de todos os métodos de uma classe em Python tem que ser o *self*. Ele representa a instância sobre a qual o método atuará.

- os atribudos de um objeto tem seu valor preservado durante todo o tempo de vida do objeto. O tempo de vida de um objeto é o tempo em que alguma variável do seu programa o referencia.

```
1  class Conta:
2      def __init__(self, numero, saldo=0):
3          self.saldo =saldo
4          self.numero = numero
5
6      def resumo(self):
7          print("CC numero: %s Saldo: %10.2f" % (self.numero, self.saldo))
8
9      def saque(self, valor):
10         if self.saldo>=valor:
11             self.saldo -= valor
12
13     def deposito(self, valor):
14         self.saldo += valor
```

Classes e objetos

Classes e objetos - exemplo - Conta bancária

- Apesar de imprescindível na **definição** de cada método, não é necessário passar o *self* como parâmetro na hora de **chamar** um método, isso é feito automaticamente no interpretador Python.

```
In [1]:  c1=Conta(1234, 100)

In [2]: c1.resumo
Out[2]: <bound method Conta.resumo of <__main__.Conta object at 0x00000294A612DA58
     >>

In [3]: c1.resumo()
Out[3]: CC numero: 1234 Saldo:      100.00

In [4]: c1.saque(50)

In [5]: c1.resumo()
Out[5]: CC numero: 1234 Saldo:       50.00

In [6]: c1.deposito(200)

In [7]: c1.resumo()
Out[7]:CC numero: 1234 Saldo:      250.00
```

Classes e objetos

- Encapsulamento:
  - O underscore _ alerta que ninguém deve modificar, nem mesmo ler, o atributo em questão;
  - Crie métodos getters e setters para realizar a interface com a classe;

```
class Conta:
    def __init__(self, numero, saldo=0):
        self.saldo =saldo
        self.numero = numero
```

```
class Conta:
    def __init__(self, numero, saldo=0):
        self._saldo = saldo
        self._numero = numero
    def get_saldo(self):
        return self._saldo
    def set_saldo(self, saldo):
        if(saldo < 0):
            print("saldo não pode ser negativo")
        else:
            self._saldo = saldo
```

Classes e objetos

- Encapsulamen
  - O underscore mesmo ler, o atributo em
  - Crie métodos om a classe;


```
class Conta:
    def __init__(sel
        self.saldo =
        self.numero
```

```
, numero, saldo=0):
 saldo
= numero
f):
saldo
f, saldo):
:
ldo não pode ser negativo")

do = saldo
```

UNIVERSIDADE
CANDIDO
MENDES

EAD