Tomada de Decisão
(execução condicional)

Exemplo motivador

Escreva uma função **testa_aprovado** que receba as duas notas de um aluno (n1 e n2), calcule e exiba a sua média. Caso o aluno esteja aprovado, exiba a mensagem 'está aprovado'

- ✓ um aluno está aprovado SOMENTE se sua média for maior ou igual a 5.0

Teste sua função chamando-a no console com vários valores, por exemplo:

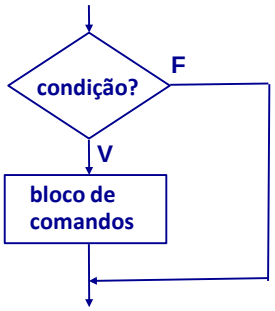
```
testa_aprovado(8.0, 7.0)
testa_aprovado(3.0, 8.5)
testa_aprovado(4.0, 5.0)
```

COMO FAZER ISSO?

Execução Condicional

Para escrever programas úteis, é necessário verificar condições para decidir o que fazer.

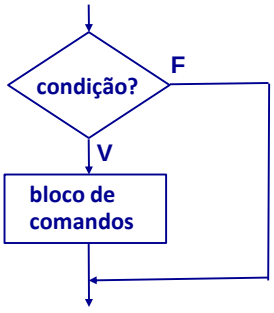
- ✓ a forma mais simples de um comando condicional é o comando **if**



3

Condicional: comando IF

A forma mais simples de um comando condicional é o comando **if**



a condição é o resultado de uma expressão

o cabeçalho termina com :

if *expressão* **:**

<instrução$_1$>

...

<instrução$_n$>

este bloco será executado se a condição for verdadeira

4

Expressões Booleanas (1/3)

Expressões booleanas produzem como resultado um valor **verdadeiro** ou **falso**.

- ✓ uma expressão booleana é geralmente construída com operadores relacionais.

Expressões Booleanas (2/3)

Expressões booleanas produzem como resultado um valor **verdadeiro** ou **falso**.

- ✓ uma expressão booleana é geralmente construída com operadores relacionais.

```
x == y          # x é igual a y ?

x != y          # x é diferente de y?
x > y           # x é maior do que y?
x < y           # x é menor do que y?
x >= y          # x é maior ou igual a y?
x <= y          # x é menor ou igual a y?
```

Expressões Booleanas (3/3)

Expressões booleanas produzem como resultado um valor **verdadeiro** ou **falso**.

- ✓ uma expressão booleana é geralmente construída com operadores relacionais.

```
x == y          # x é igual a y ?

x != y          # x é diferente de y?
x > y           # x é maior do que y?
x < y           # x é menor do que y?
x >= y          # x é maior ou igual a y?
x <= y          # x é menor ou igual a y?
```

Atenção!
= é o operador de atribuição!

O que será exibido?

```
x = -10
if x < 0:
  print("O numero negativo ", x, "nao e valido aqui.")

print("Continuemos.")
```

O que será exibido? Bloco do if

```
x = -10
if x < 0:
  print("O numero negativo ", x, "nao e valido aqui.")

print("Continuemos.")``
```

```
O numero negativo -10 nao e valido aqui.
Continuemos.
```

- ✓ Se a condição for verdadeira, o bloco de comandos é executado e, a seguir, o que vem depois do **if**.

E agora?

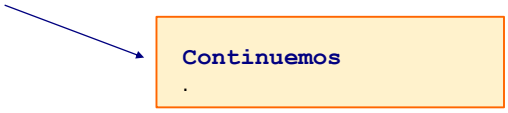
```
x = 10
if x < 0:
  print("O numero negativo ", x, "nao e valido aqui.")

print("Continuemos.")
```

E agora? Condição falsa!

```
x = 10
if x < 0:
  print("O numero negativo ", x, "nao e valido aqui.")

print("Continuemos.")
```



- ✓ Se a condição for falsa, o bloco de comandos **não é** executado e a execução continua imediatamente com o que vem depois do **if**.

11

Expressões e operandos (1/5)

Os operandos de uma expressão booleana podem ser resultados de expressões:

12

Expressões e operandos (2/5)

Os operandos de uma expressão booleana podem ser resultados de expressões:

```
if (x % 2) == 0:
```

→ x é múltiplo de 2?

Expressões e operandos (3/5)

Os operandos de uma expressão booleana podem ser resultados de expressões:

```
if (x % 2) == 0:
```

→ x é múltiplo de 2?

```
if c <= (a + b):
```

→ c é menor ou igual à soma de a e b?

Expressões e operandos (4/5)

Os operandos de uma expressão booleana podem ser resultados de expressões:

```
if (x % 2) == 0:
```
→ x é múltiplo de 2?

```
if c <= (a + b):
```
→ c é menor ou igual à soma de a e b?

```
nome = input("Nome: ")
if len(nome) > 30:
```
→ o tamanho da string é maior que 30?

Expressões e operandos (5/5)

Os operandos de uma expressão booleana podem ser resultados de expressões:

```
if (x % 2) == 0:
```
→ x é múltiplo de 2?

```
if c <= (a + b):
```
→ c é menor ou igual à soma de a e b?

```
nome = input("Nome: ")
if len(nome) > 30:
```
→ o tamanho da string é maior que 30?

```
if (pdir - t.xcor()) < 20:
```
→ a distância para a parede é menor que 20?

Comparando floats

Valores do tipo float são aproximações!

```
x =  1.1 / 10
print(x)
```

→ `0.11000000000000001`

Comparando floats: aproximação

Valores do tipo float são aproximações!

```
x =  1.1 / 10
print(x)
```

→ `0.11000000000000001`

Nunca devemos testar se valores do tipo float são iguais, mas sim se são suficientemente próximos.

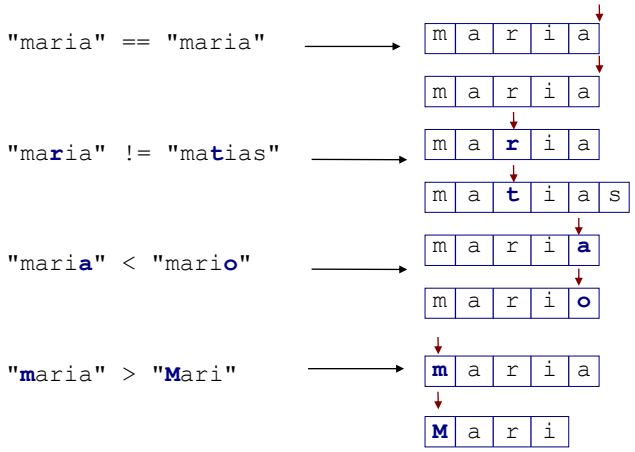
```
# para cada problema, existe um valor adequado
delta = 0.0001

if abs(x-y) < delta:
   ...
```

← retorna valor absoluto do argumento

Comparando strings

Strings são comparadas lexicograficamente (em “ordem alfabética”), caractere a caractere:



Mãos na massa: média do aluno

Escreva uma função **testa_aprovado** que receba as duas notas de um aluno (n1 e n2), calcule e exiba a sua média. Caso o aluno esteja aprovado, exiba a mensagem 'está aprovado'.

- ✓ um aluno está aprovado se sua média for maior ou igual a 5.0

Teste sua função chamando-a na console com vários valores, por exemplo:

```
testa_aprovado(8.0, 7.0)
testa_aprovado(3.0, 8.5)
testa_aprovado(4.0, 5.0)
```

Uma solução

```
def testa_aprovado(n1, n2):

  media = (n1 + n2) / 2
  print("média = ", media)

  if media >= 5.0:
    print("está aprovado")

  return
```


Testando várias condições

Modifique a sua função **testa_aprovado** para aplicar um novo critério de aprovação.

- ✓ um aluno está aprovado se sua média for maior ou igual a 5.0 e nenhuma das notas for menor que 3.0

Uma solução (provisória!)

```
def testa_aprovado(n1, n2):

  media = (n1 + n2) / 2
  print("média = ", media)

  if media >= 5.0:
    if n1 >= 3.0:
       if n2 >= 3.0:
            print("está aprovado")

  return
```


Operadores lógicos

Expressões booleanas também podem ser formadas com operadores lógicos:

not: NÃO (negação)

and: E (conjunção)

or: OU (disjunção)

Operadores lógicos: and

<operando1> and <operando2>

operando 1	operando 2	resultado
falso	falso	falso
falso	verdadeiro	falso
verdadeiro	falso	falso
verdadeiro	verdadeiro	**verdadeiro**

Operadores lógicos: or

<operando1> or <operando2>

operando 1	operando 2	resultado
falso	falso	falso
falso	verdadeiro	**verdadeiro**
verdadeiro	falso	**verdadeiro**
verdadeiro	verdadeiro	**verdadeiro**

Operadores lógicos: not

not <operando>

operando	resultado
falso	**verdadeiro**
verdadeiro	falso

Operadores lógicos: exemplo and

```
x > 0 and x < 10
```

→ verdadeiro **somente** se as duas condiçoes são verdadeiras

→ x é maior do que 0 **E TAMBÉM** x é menor do que 10

Operadores lógicos: exemplo or

```
x > 0 and x < 10
```

→ verdadeiro **somente** se as duas condiçoes são verdadeiras

→ x é maior do que 0 **E TAMBÉM** x é menor do que 10

```
n % 2 == 0 or n % 3 == 0
```

→ verdadeiro se **QUALQUER UMA** das condições é verdadeira (ou ambas):

→ n é múltiplo de 2 **OU** n é múltiplo de 3

Operadores lógicos: exemplo not

```
x > 0 and x < 10
```

→ verdadeiro **somente** se as duas condiçoes são verdadeiras

→ x é maior do que 0 **E TAMBÉM** x é menor do que 10

```
n % 2 == 0 or n % 3 == 0
```

→ verdadeiro se **QUALQUER UMA** das condições é verdadeira (ou ambas):

→ n é múltiplo de 2 **OU** n é múltiplo de 3

```
not (x < y)
```

→ verdadeiro se a condição é falsa

→ x **NÃO É** menor que y

Atenção: exemplo com or! (1/3)

Suponha que queremos testar se uma variável **n** tem valor igual a 5 ou valor igual a 6

- ✓ em Português, poderíamos dizer: “se n é igual a 5 ou 6”

Atenção: exemplo com or! (2/3)

Suponha que queremos testar se uma variável **n** tem valor igual a 5 ou valor igual a 6

- ✓ em Português, poderíamos dizer: “se n é igual a 5 ou 6”

Se traduzirmos este teste para Python como

```
if n == 5 or 6:
```

a tradução não estará correta, pois precisamos deixar explícitas todas as condições.

Atenção: exemplo com or! (3/3)

Suponha que queremos testar se uma variável **n** tem valor igual a 5 ou valor igual a 6

- ✓ em Português, poderíamos dizer: “se n é igual a 5 ou 6”

Se traduzirmos este teste para Python como

```
if n == 5 or 6:
```

a tradução não estará correta, pois precisamos deixar explícitas todas as condições.

A forma correta de escrever esse teste em Python é

```
if n == 5 or n == 6:
```


Voltando ao exemplo motivador

A função **testa_aprovado** recebe as duas notas de um aluno (n1 e n2), calcula e exibe a sua média. Caso o aluno esteja aprovado, exibe a mensagem 'está aprovado'.

```
def testa_aprovado(n1, n2):

  media = (n1 + n2) / 2
  print("média = ", media)

  if media >= 5.0 and n1 >= 3.0 and n2 >= 3.0:
    print("está aprovado")

  return
```

Mãos na massa: prova final!

Escreva uma função que receba as duas notas de um aluno (n1 e n2), calcule e exiba a sua média. Caso o aluno **não esteja aprovado**, exiba a mensagem 'está na final'.

- ✓ lembre-se de que um aluno está **aprovado** se sua média for maior ou igual a 5.0 e nenhuma das notas for menor que 3.0

Uma solução

```
def testa_final(n1, n2):

  media = (n1 + n2) / 2
  print("média = ", media)


  if media < 5.0 or n1 < 3.0 or n2 < 3.0:

    print("está na final!")

  return
```

Mais uma mensagem possível

```
def testa_final(n1, n2):
  media = (n1 + n2) / 2
  print("média = ", media)

  if media < 5.0 or n1 < 3.0 or n2 < 3.0:
    print("está na final!")

  return
```

Modifique a sua função para que ela exiba, também, a mensagem 'precisa de muita atenção!', para alertar os alunos que estão em prova final e que têm pelos menos uma das notas menor do que 2.0 e a segunda nota menor do que a primeira.

37

Uma solução

```
def testa_final(n1, n2):

  media = (n1 + n2) / 2
  print("média = ", media)


  if media < 5.0 or n1 < 3.0 or n2 < 3.0:
    print("está na final!")
    if  (n1<2.0 or n2< 2.0) and n2 < n1:
        print("precisa de muita atenção!")

  return
```

38

Ordem de avaliação de operadores

① **
② * , / , // , %
③ + , -
④ == , != , <= , >= , > , <
⑤ not
⑥ and
⑦ or

① ② ③ → aritméticos
④ → relacionais
⑤ ⑥ ⑦ → lógicos

Avaliação de operadores (1/4)

```
if x > y + 1 and y > z
```

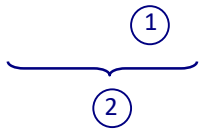
① (y + 1)

① **
② * , / , // , %
③ + , -
④ == , != , <= , >= , > , <
⑤ not
⑥ and
⑦ or

① ② ③ → aritméticos
④ → relacionais
⑤ ⑥ ⑦ → lógicos

Avaliação de operadores (2/4)

```
if x > y + 1 and y > z
```



① **
② *, /, //, %
③ +, -
④ ==, !=, <=, >=, >, <
⑤ not
⑥ and
⑦ or

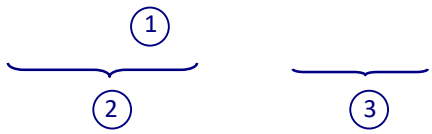
aritméticos

relacionais

lógicos

Avaliação de operadores (3/4)

```
if x > y + 1 and y > z
```



① **
② *, /, //, %
③ +, -
④ ==, !=, <=, >=, >, <
⑤ not
⑥ and
⑦ or

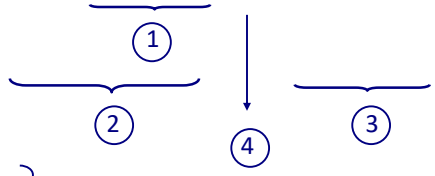
aritméticos

relacionais

lógicos

Avaliação de operadores (4/4)

```
if x > y + 1 and y > z
```



① **
② * , / , // , %
③ + , -
④ == , != , <= , >= , > , <
⑤ not
⑥ and
⑦ or

aritméticos

relacionais

lógicos

Valores Booleanos

Em Python, existem dois valores booleanos (tipo bool), True e False.

- ✓ o resultado da avaliação de uma expressão booleana é um valor booleano

```
print(5 == 5)
print(2 > 5)
```

→ `True`

→ `False`

Valores Booleanos vs strings

Em Python, existem dois valores booleanos (tipo bool), True e False.

- ✓ o resultado da avaliação de uma expressão booleana é um valor booleano

```
print(5 == 5)
print(2 > 5)
```

→

```
True
False
```

Valores booleanos não são strings!

```
print(type(True))
print(type("True"))
```

→

```
<class 'bool'>
<class 'str'>
```


Funções Booleanas

Como o resultado de uma expressão booleana é um valor booleano, podemos escrever uma função que retorna um valor booleano.

```
def testa_aprovado(n1, n2):
    media = (n1 + n2) / 2
    return media >= 5.0 and n1 >= 3.0 and n2 >= 3.0
```

a função vai retornar o resultado da expressão booleana

Funções Booleanas e condicionais

Como o resultado de uma expressão booleana é um valor booleano, podemos escrever uma função que retorna um valor booleano.

a função vai retornar o resultado da expressão booleana

```
def testa_aprovado(n1, n2):
  media = (n1 + n2) / 2

  return media >= 5.0 and n1 >= 3.0 and n2 >= 3.0

nota1 = float(input("primeira nota: ")
nota2 = float(input("segunda nota: ")

if testa_aprovado(nota1, nota2):
  print("está aprovado!")
```

a condição é o valor retornado pela função!

47

Mãos na massa: equilátero!

Escreva (e teste) uma função booleana **equilatero** que recebe os três lados de um triângulo e retorna **True**, se eles formam um triângulo equilátero, ou **False**, caso contrário.

48

Equilátero: definição inicial

Escreva (e teste) uma função booleana **equilatero** que recebe os três lados de um triângulo e retorna **True**, se eles formam um triângulo equilátero, ou **False**, caso contrário.

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)
```


Triângulo isósceles não equilátero

Escreva agora uma função booleana chamada **isosceles_nao_equilatero** que recebe os três lados de um triângulo e retorna **True**, se eles formam um triângulo isósceles não equilátero, ou **False**, caso contrário.

- ✓ um triângulo é isósceles se tem dois lados iguais
- ✓ use sua função **equilatero** para testar se o triângulo é equilátero

Uma solução correta (1/5)

```
def isosceles_nao_equilatero(a,b,c):
  return (a==b or b==c or a==c) and not equilatero(a,b,c)
```

atenção para os parênteses para controlar a ordem de avaliação!

① ==, !=, <=, >=, >, <
② not
③ and
④ or

51

Uma solução correta (2/5)

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return (a==b or b==c or a==c) and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

False

```
(3==3 or 3==3 or 3==3) and not equilatero(3,3,3)
```

True

52

Uma solução correta (3/5)

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return (a==b or b==c or a==c) and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

False

```
(3==3 or 3==3 or 3==3) and not equilatero(3,3,3)
```

True **True**

Uma solução correta (4/5)

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return (a==b or b==c or a==c) and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

False

```
(3==3 or 3==3 or 3==3) and not equilatero(3,3,3)
```

True **True**

False

Uma solução correta (5/5)

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return (a==b or b==c or a==c) and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

False

```
(3==3 or 3==3 or 3==3) and not equilatero(3,3,3)
```

True **True** **False** **False**

55

Uma solução incorreta (1/4)

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return a==b or b==c or a==c and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

True

```
3==3 or 3==3 or 3==3 and not equilatero(3,3,3)
```

True **True** **True** **True**

56

Uma solução incorreta (2/4)

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return a==b or b==c or a==c and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

→ **True**

```
3==3 or 3==3 or 3==3 and not equilatero(3,3,3)
```

True **True** **True** **True** **False**

Uma solução incorreta (3/4)

```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return a==b or b==c or a==c and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

→ **True**

```
3==3 or 3==3 or 3==3 and not equilatero(3,3,3)
```

True **True** **True** **True** **False**

False

Uma solução incorreta (4/4)

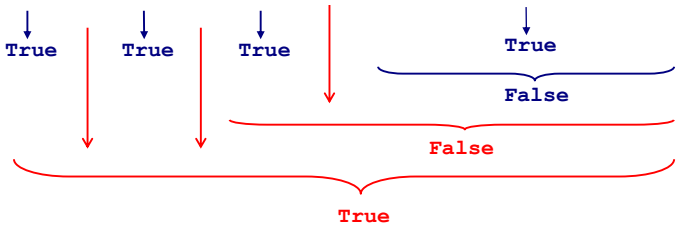
```
def equilatero(a,b,c):
  return (a==b) and (b==c) and (a==c)

def isosceles_nao_equilatero(a,b,c):
  return a==b or b==c or a==c and not equilatero(a,b,c)

>>> isosceles_nao_equilatero(3,3,3)
```

True

```
3==3 or 3==3 or 3==3 and not equilatero(3,3,3)
```



Exercício: ano bissexto

Um ano é bissexto se ele é múltiplo de 4

e

não é o ano de término de um século (isto é, não é múltiplo de 100)

ou

é divisível por 400

Escreva (e teste) uma função **testa_bissexto** que peça ao usuário que informe um ano (convertendo a entrada para um inteiro) e, se o ano for bissexto, imprima uma mensagem.

Bissexto: uma solução

```
def testa_bissexto():

  ano = int(input("Qual o ano? "))

   # se ano é múltiplo de 4 e (não é final de século ou é múltiplo de 400)

  if ano % 4 == 0 and
     (ano % 100 != 0 or ano % 400 == 0):

    print("é bissexto")

  return
```


Mãos na massa: painel do carro!

O computador de bordo de um carro oferece uma função de consulta, que o motorista utiliza para saber se é necessário reabastecer.

Essa função pede ao motorista a distância a ser percorrida e obtém, de outras funções disponíveis no sistema do carro, a quantidade atual de litros no tanque e o consumo do carro no momento (em km/l).

Com esses dados, a função calcula a quantidade necessária de litros para percorrer a distância desejada.

Se a quantidade de litros no tanque não for suficiente para alcançar a distância desejada, o motorista é avisado.

Painel do carro: ideia

```
# simula a obtenção do volume atual no tanque
def obtem_litros_tanque():
  return int(input("volume atual no tanque: "))

# simula a obtenção do consumo atual do carro
def obtem_consumo_atual():
  return int(input("consumo atual (km/l): "))

# verifica a necessidade de reabastecer
def alerta_abastecer(distancia):
  distancia = int(input("Qual a distancia? "))
  litros_tanque = obtem_litros_tanque()
  consumo = obtem_consumo_atual()

  # Se os litros no tanque não forem suficientes
  # emitir uma mensagem

  return
```

63

Painel do carro: uma solução

```
# verifica a necessidade de reabastecer
def alerta_abastecer(distancia):
  distancia = int(input("Qual a distancia? "))
  litros_tanque = obtem_litros_tanque()
  consumo = obtem_consumo_atual()

  # Se os litros no tanque não forem suficientes
  # emitir uma mensagem
  if distancia/consumo > litros_tanque:
     print("Necessario reabastecer!")

  return
```

64

Tomada de Decisão
(execução alternativa)

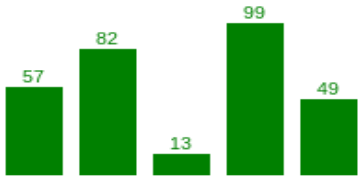
Diagrama de Barras

Vamos agora usar a tartaruga para desenhar um diagrama de barras:

- ✓ usaremos uma sequência de valores para as alturas das barras
- ✓ para cada altura, desenharemos um retângulo com essa altura e uma largura fixa;
- ✓ acima do retângulo, escreveremos o valor (altura) associado.

Exemplo de diagrama

Para alturas com valores 57, 82, 13, 99 e 49, nosso diagrama deverá ficar assim:



Desenhando um retângulo

```
import turtle

def retangulo(t, altura, largura, cor):
    t.setheading(90)          # aponta para cima
    t.color(cor)
    t.begin_fill()             # iniciar preenchimento
    t.forward(altura)
    t.right(90)
    t.forward(largura)
    t.right(90)
    t.forward(altura)
    t.right(90)
    t.forward(largura)
    t.right(90)
    t.end_fill()               # terminar preenchimento
    return
```

Desenhando uma barra

```
def barra(t, posX, posY, largura, altura, cor):
    # posiciona e desenha o retangulo
    t.goto(posX, posY)
    t.down()
    retangulo(t, altura, largura, cor)
    # posiciona e escreve o valor
    t.goto(posX+largura/3,posY + altura)
    t.write(str(altura))
    t.up()
    t.hideturtle()
    return
```


Diagrama de barras (1/2)

```
import turtle
def retangulo(t, altura, largura, cor):
  <...>
def barra(t, posX, posY, largura, altura):
  <...>
diagrama = turtle.Turtle()
diagrama.up()
posX = 0
posY = 0
largura = 30 57, 82, 13, 99, 49
cor = "green"
altura = 57
barra(diagrama, posX, posY, largura, altura, cor)
posX = posX + largura + 10
…   …   …
```

Diagrama de barras (2/2)

```
… … …
altura = 82
barra(diagrama, posX, posY, largura, altura, cor)
posX = posX + largura + 10
altura = 13
barra(diagrama, posX, posY, largura, altura, cor)
posX = posX + largura + 10
altura = 99
barra(diagrama, posX, posY, largura, altura, cor)
posX = posX + largura + 10
altura = 49
barra(diagrama, posX, posY, largura, altura, cor)
posX = posX + largura + 10
```

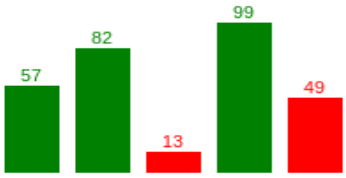

Mudando a cor das barras

E se quiséssemos desenhar um diagrama com cores diferentes para as barras? Usando as seguintes regras:

- se o valor da altura é menor que 50, a cor da barra deverá ser vermelha,
- caso contrário, a cor da barra deverá ser verde.

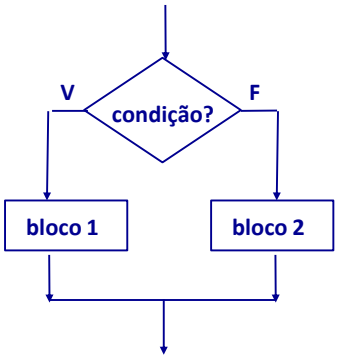
Novo diagrama

Para alturas com valores 57, 82, 13, 99 e 49, nosso novo diagrama deverá ficar assim:



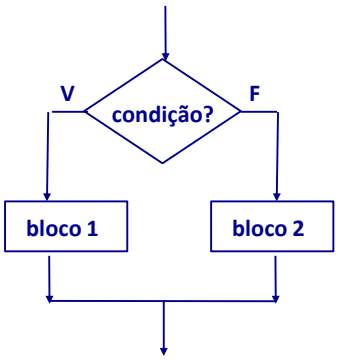
Execução alternativa

Uma outra forma de tomada de decisão é a "execução alternativa", quando escolhemos uma ação entre duas ações possíveis.



Execução alternativa: else

Uma outra forma de tomada de decisão é a "execução alternativa", quando escolhemos uma ação entre duas ações possíveis.



executado se a condição é **verdadeira**

```
if expressão :
    <bloco de comandos 1>
else:
    <bloco de comandos 2>
```

executado se a condição é **falsa**

Voltando às barras

```
def define_cor(altura):
  if altura < 50:
    cor = "red"
  else:
    cor = "green"
  return cor
```

é executado se condição é verdadeira

é executado se condição é falsa

```
diagrama = turtle.Turtle()
<...>
cor = define_cor(altura)
barra(diagrama, posX, posY, largura, altura, cor)
posX = posX + largura + 10
altura = 82
… …
```

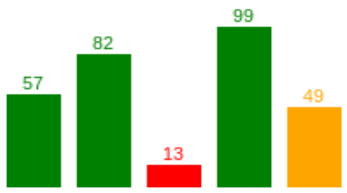
Mais cores para as barras

E se quiséssemos desenhar um diagrama com mais cores para as barras? Usando as seguintes regras:

- se o valor da altura é menor que 30, a cor da barra deverá ser vermelha;
- se o valor da altura é maior ou igual a 30, mas é menor que 50, a cor da barra deverá ser laranja;
- se o valor da altura é maior ou igual a 50, a cor da barra deverá ser verde.

Diagrama com mais cores

Para alturas 57, 82, 13, 99 e 49, nosso novo diagrama deverá ficar assim:



Condicionais aninhados

```
def define_cor(altura):

  if altura < 30:
    cor = "red"
  else:
    if altura < 50:
      cor = "orange"
    else:
      cor = "green"
  return cor
```

podemos aninhar comandos condicionais dentro de outros

Condicionais encadeados

Quando há mais de duas possibilidades, em Python podemos também usar condicionais encadeados.

```
def define_cor(altura):

  if altura < 30:
    cor = "red"
  elif altura < 50:
    cor = "orange"
  else:
    cor = "green"

  return cor
```

executado se nenhuma das condições é verdadeira

Não há limite para o número de **elif** mas só pode haver um único **else**, no final !

Exercícios if e else (1/2)

1) Escreva (e teste!) uma função, chamada **maior**, que receba dois números e retorne o maior deles.

2) Escreva (e teste!) uma função, chamada **menor**, que compara o valor dos seus três parâmetros inteiros (a,b,c) e retorna o menor deles.

3) Dados três gravetos, só podemos construir um triângulo com eles se nenhum dos gravetos tem tamanho maior que a soma dos tamanhos dos outros dois.

   - ✓ Escreva (e teste!) uma função, chamada **formam_triangulo**, que receba o comprimento de três gravetos e exiba 'Sim' se eles podem formar um triângulo, ou 'Não', caso contrário.

Exercícios if e else (2/2)

4) Escreva uma função, chamada **resultado_aluno**, que receba as duas notas de um aluno (n1 e n2), calcule a sua média e informe se o aluno está aprovado, reprovado ou deve fazer prova final.

   - ✓ um aluno está aprovado se sua média for maior ou igual a 5.0 e as duas notas forem maiores ou iguais a 3.0;
   - ✓ um aluno está reprovado se a média for inferior a 3.0;
   - ✓ em qualquer outra situação, o aluno está em prova final. Neste caso, a função deve exibir qual a nota mínima na prova final para que o aluno seja aprovado.
     - ✓ a prova final deve substituir a menor nota e a nova média deve ser maior ou igual a 5.0.

Algumas soluções: maior e menor

```
def maior(a,b):
  if a > b:
    return a
  else:
    return b

def menor(a,b,c):
  if a < b:
    if a < c:
      return a
    else:
      return c
  else:
    if b < c:
      return b
    else:
       return c
```


Solução para formam_triangulo

```
def formam_triangulo(a,b,c):

  # se lado a é menor que a soma de b e c ou
  #    lado b é menor que a soma de a e c ou
  #    lado c é menor que a soma de a e b
  #    → imprime "Sim"

  if a < (b+c) and b < (a+c) and c < (a+b):
    print("Sim")

  # caso contrário imprime "Não"
  else:
    print("Nao")
```

Solução para resultado_aluno

```
def resultado_aluno(n1,n2):
  media = (n1+n2)/2
  if media >= 5.0:
    print("aprovado")
  elif media < 3.0:
    print("reprovado")
  else:
    if n1 < n2:
      final = 10.0 - n2
    else:
      final = 10.0 - n1
    print("em final, precisa tirar", final)
```


Exercício da OBI: avião de papel

A Diretora de uma escola organizou um campeonato de aviões de papel. Cada aluno participante receberá um certo número de folhas de papel especial para fazer os seus aviões. Esse número será decidido pelos juízes do campeonato.

Considere, por exemplo, que a Diretora comprou 100 folhas de papel, e que há 33 competidores.

- ✓ se os juízes decidirem que cada competidor receberá 3 folhas, a quantidade comprada é suficiente.
- ✓ se os juízes decidirem que cada competidor receberá 4 folhas, a quantidade comprada não é suficiente.

OBI: quantidade de folhas

Escreva uma função, chamada **verifica_quantidade**, que, dados o número de competidores, o número de folhas compradas e o número de folhas que cada competidor deve receber, exiba uma mensagem indicando se o número de folhas compradas é suficiente ou não.

Exemplos:

- ✓ Entrada: 10,100,10 → Saída: 'Suficiente'
- ✓ Entrada: 10,90,10 → Saída: 'Insuficiente'
- ✓ Entrada: 5,40,2 → Saída: 'Suficiente'

Avião de papel: uma solução

```
def verifica_quantidade(alunos, folhas, folhas_por_aluno):

  #quantidade necessaria
  quant = folhas_por_aluno * alunos

  if folhas < quant:
    print("Insuficiente")
  else:
    print("Suficiente")
```

Outro exercício da OBI: futebol

Dois times de futebol, Cormengo e Flaminthians, participam de um campeonato em que cada vitória conta 3 pontos e cada empate conta 1 ponto.

Os critérios para verificar qual o time está mais bem classificado são:

- está mais bem classificado o time que tenha mais pontos;
- em caso de empate no número de pontos, está mais bem classificado o time que tiver maior saldo de gols;
- se o número de pontos e o saldo de gols forem os mesmos, eles estão empatados.

OBI: melhor time

Escreva uma função, chamada **melhor_time**, que retorne o nome do time mais bem classificado, ou 'Empate', se eles estão empatados.

Os parâmetros de entrada da função são seis números inteiros: cv, ce, cs, fv, fe e fs, respectivamente, o número de vitórias, empates e saldo de gols do Cormengo, o número de vitórias, empates e saldo de gols do Flaminthians.

Exemplos:

- Entrada: 10, 5, 18, 11, 2, 18 → Saída: 'Empate'
- Entrada: 10, 5, 18, 11, 1, 18 → Saída: 'Cormengo'
- Entrada: 9, 5, -1, 10, 2, 10 → Saída: 'Flaminthias'

Times de futebol: uma solução

```
def melhor_time(cv,ce,cs,fv,fe,fs):

  notac = cv*3 + ce
  notaf = fv*3 + fe

  if notac > notaf:
    return "Carmengo"
  elif notaf > notac:
    return "Flaminthias"
  elif cs > fs:
    return "Carmengo"
  elif fs > cs:
    return "Flaminthias"
  else:
    return "Empate"
```

91

Exercício: abastecimento de água

Para resolver o abastecimento de água de uma cidade, seu governo avalia a largura (em metros), a profundidade (em metros) e a vazão (em l/s) de um rio da região. A nota da avaliação do rio é calculada conforme as regras abaixo:

- largura inferior a 15m: **nota zero**;
- profundidade inferior a 6m: **nota zero**;
- largura entre 15m e 30m (inclusive) e profundidade maior ou igual a 6m:
  - vazão inferior a 5000 l/s: **nota = 5 + vazão/2700**
  - vazão maior ou igual a 5000 l/s: **nota = 5 + vazão/2500**
- largura superior a 30m e profundidade maior ou igual a 6m:
  - **nota = 5 + vazao/2000**

92

Água: parte 1 - vazão

Escreva uma função, chamada **nota_rio**, que receba a largura (em m), a profundidade (em m) e a velocidade do rio (em m/s), e retorne a sua nota, conforme as regras dadas. Para calcular a vazão do rio, a função deve usar a equação:

vazao = largura X profundidade X velocidade

Parte 1: uma solução

```
def nota_rio(largura, profundidade, velocidade):
  if largura < 15:
    return 0
  elif profundidade < 6:
    return 0
  else:
    vazao = largura * profundidade * velocidade
    if largura <= 30:
        if vazao < 5000:
          return 5 + vazao/2700
        else:
          return 5 + vazao/2500
    else:
      return 5 + vazao/2000
```

Água: parte 2 - avalia

Escreva agora uma função, chamada **avalia_rio**, que receba a largura, a profundidade e a velocidade do rio e, usando a função **nota_rio**, faça o seguinte:

- ✓ exiba a nota do rio;
- ✓ exiba a mensagem 'Alternativa viável', se essa nota for maior ou igual a 7.0.

Parte 2: uma solução

```
def nota_rio(largura, profundidade, velocidade):
  if largura < 15:
    return 0
  elif profundidade < 6:
    return 0
  else:
    vazao = largura * profundidade * velocidade
    if largura <= 30:
        if vazao < 5000:
          return 5 + vazao/2700
        else:
          return 5 + vazao/2500
    else:
      return 5 + vazao/2000

def avalia_rio(largura, profundidade, velocidade):
  nota = nota_rio(largura, profundidade, velocidade)
  print("Nota do rio:",nota)
  if nota >= 7.0:
    print ("Alternativa viavel")
  return
```

Operador *in* e Caracteres

Pertinência: operador in

O operador booleano **in** verifica se uma dada string pertence (ou está presente ou contido) em outra string. Analogamente: o **in** verifica se uma string é uma substring de outra string (*sequência x ⊂ sequência y?)*

```
"gosto" in "Eu gosto de programar!"  ⟶  True
"odeio" in "Eu gosto de programar!"  ⟶  False
"y" in "Python"                      ⟶  True
"i" in "Python                       ⟶  False
```

Testa a ausência de uma substring usando **not.**

```
"odeio" not in "Eu gosto de programar!"  ⟶  True
```

Obs. A string vazia é uma substring de qualquer string.

```
"" in "Python"  ⟶  True
```

Caracteres e strings

- **Quem são os caracteres?** Letras: ‘a’, ‘d’, ‘F’, ‘G’
  Símbolos: ‘%’, ‘@’, ‘:’, ‘?’
  Dígitos: ‘0’, ‘3’, ‘6’,’9’
  Símbolo Especiais: ‘\n’,’\t’,’\b’,’\r’

  “símbolos que normalmente encontra-se num teclado de um computador".

- Em Python, um caractere é uma string de comprimento 1.

COMO O CARACTERE É REPRESENTADO INTERNAMENTE?

Para serem processados pelo computador, os caracteres devem ser representados por números.

– **tabela de códigos: nº ←→ caracter**
  define correspondência entre caracteres e códigos numéricos

Exemplos: ASCII: 256 caracteres diferentes (Python 2.x)
Unicode UTF-8: 65536 caracteres diferentes (no mínimo) (Python 3.x)

99

ASCII vs UTF-8

Least Significant Character \ Most Significant Character	0	1	2	3	4	5	6	7
Hex	0	1	2	3	4	5	6	7
0	NUL	DLE	Space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	Bell	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

U+0000	000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F
0	NUL	DLE	SP	0	@	P	`	p	PAD	DCS	NBSP	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q	HOP	PU1	¡	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r	BPH	PU2	¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s	NBH	STS	£	³	Ã	Ó	ã	ó
4	EOT	DC4	$	4	D	T	d	t	IND	CCH	¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u	NEL	MW	¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	SSA	SPA	¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w	ESA	EPA	§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x	HTS	SOS	¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y	HTJ	SGCI	©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z	VTS	SCI	ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{	PLD	CSI	«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l	|	PLU	ST	¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}	RI	OSC	SHY	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~	SS2	PM	®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL	SS3	APC	¯	¿	Ï	ß	ï	ÿ

100

Funções ord e chr

A função ord converte um caractere para o código numérico correspondente na codificação Unicode.

- ✓ nessa codificação, os códigos das letras seguem a "ordem alfabética" e, por convenção, os números das minúsculas são maiores do que os das maiúsculas.

```
print(ord('A'))                    → 65
print(ord('B'))                    → 66
print(ord('B')-ord('A'))           → 1
print(ord('a')                     → 97
```

A função chr é a inversa de ord. Ela recebe um código numérico e retorna o caractere correspondente na codificação Unicode.

```
print(chr(65))                     → A
print(chr(66))                     → B
```

Caracteres: mão na massa! (1/2)

1. Escreva uma função que receba um caractere e, caso o caractere seja uma letra minúscula, retorna a letra maiúscula correspondente. Caso contrário, retorna o caractere.
2. Escreva uma função que receba um caractere e, caso o caractere seja uma letra maiúscula, retorna a letra minúscula correspondente. Caso contrário, retorna o caractere.

Caracteres: mão na massa! (2/2)

3. Escreva uma função que receba uma string X e retorne uma string Y criada do seguinte modo:
   - O caractere '-' pertence a X – retorne a segunda metade de X + '&'+ primeira metade de X;
   - O caractere '-' não pertence a X – retorne primeira metade de X + '-' + segunda metade de X;
   - O 1º caractere de Y é uma letra minúscula – torne-a maiúscula;
   - O último caractere de Y é uma letra maiúscula – torne-a minúscula.
4. Escreva uma função, chamada **ehDoEstado**, que receba a placa de um carro e um dos prefixos de um estado. Esta função retorna a parte numérica, se a placa for do estado, ou 0, caso contrário.
   - ✓ Por exemplo, as placas do Rio de Janeiro variam de KMF0001 a LVE9999. Logo, a chamada ehDoEstado('LRS3232', 'LRS') deve retornar 3232.

103

Possíveis Soluções: 1 e 2

1)

```
def paraMaiuscula(c):
    if c >= 'a' and c <= 'z':
        return chr(ord('A') + (ord(c) - ord('a')))
    else:
        return c
```

→ **Soluções usam a ordem relativa das letras no alfabeto, mesmo desconhecendo os códigos numéricos!**

2)

```
def paraMinuscula(c):
    if c >= 'A' and c <= 'Z':
        return chr(ord('a') + (ord(c) - ord('A')))
    else:
        return c
```

104

Possível solução: 3

```
3)
def novaString(strX):
   meio = len(strX)//2
   if '-' in strX:
      strY= strX[meio:] + '&' +
   strX[:meio]
   else:
       strY= strX[:meio] + '-' +
   strX[meio:]

   prim = paraMaiuscula(strY[0])
   ult =  paraMinuscula(strY[-1])
   return prim+ strY[1:-1]+ ult
```


Possível solução: 4

```
4)
def ehDoEstado(placa, prefixo):
   if prefixo in placa:
      return int(placa[3:])
   else:
      return 0
```