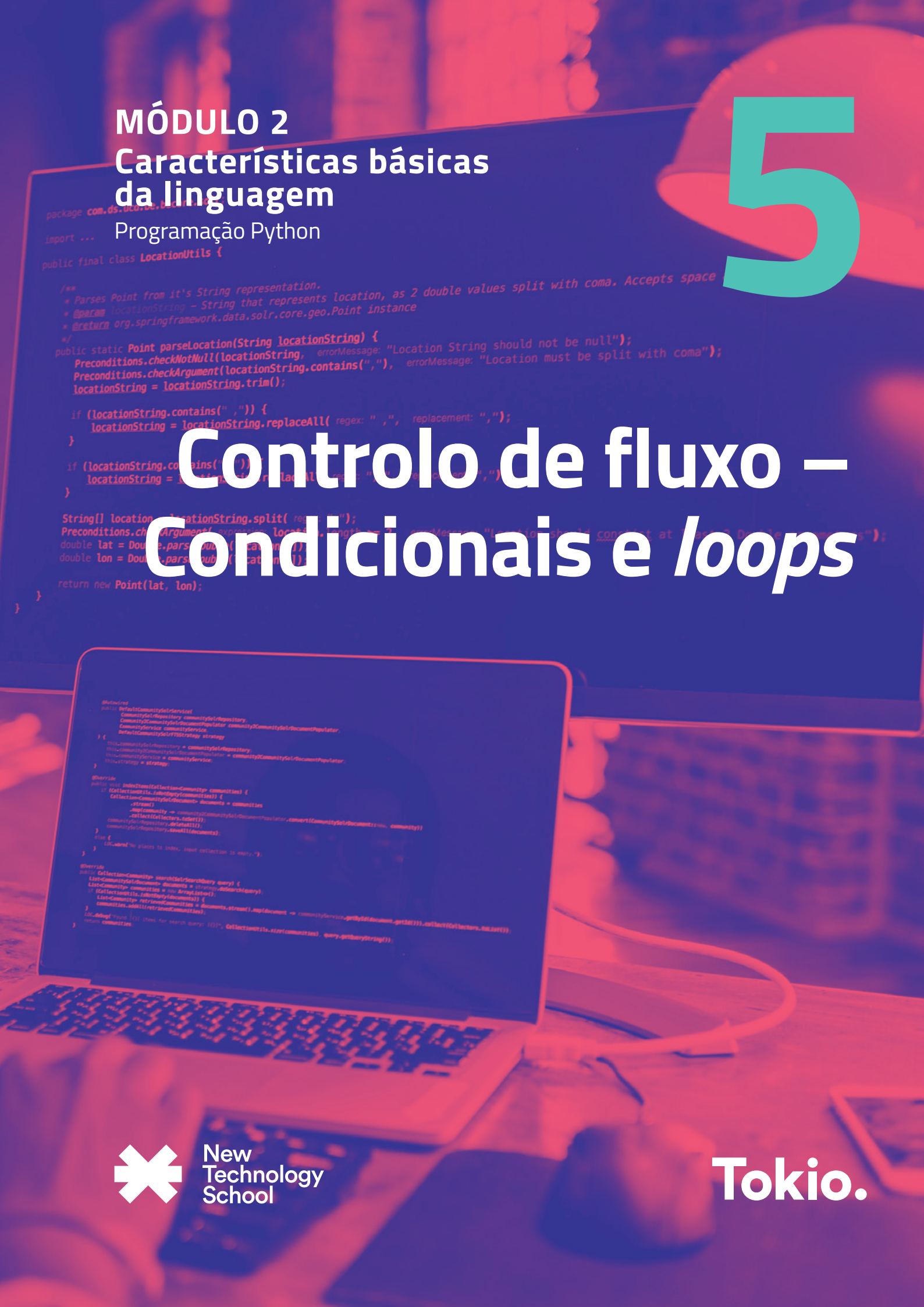
MÓDULO 2

Características básicas da linguagem

Programação Python

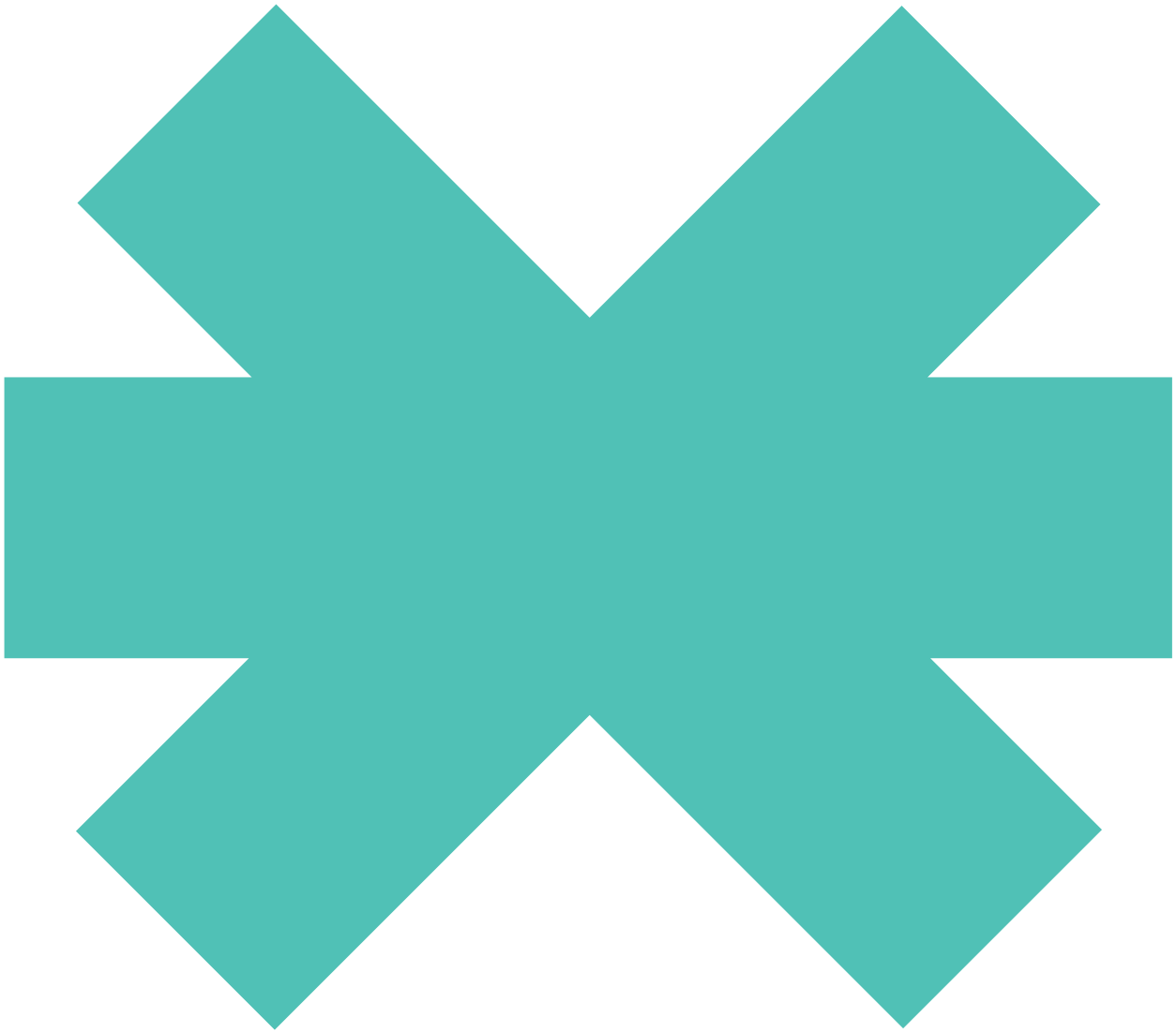
5

Controlo de fluxo – Condicionais e *loops*



New Technology School

Tokio.



5 Controlo de fluxo – Condicionais e *loops*

Sumário

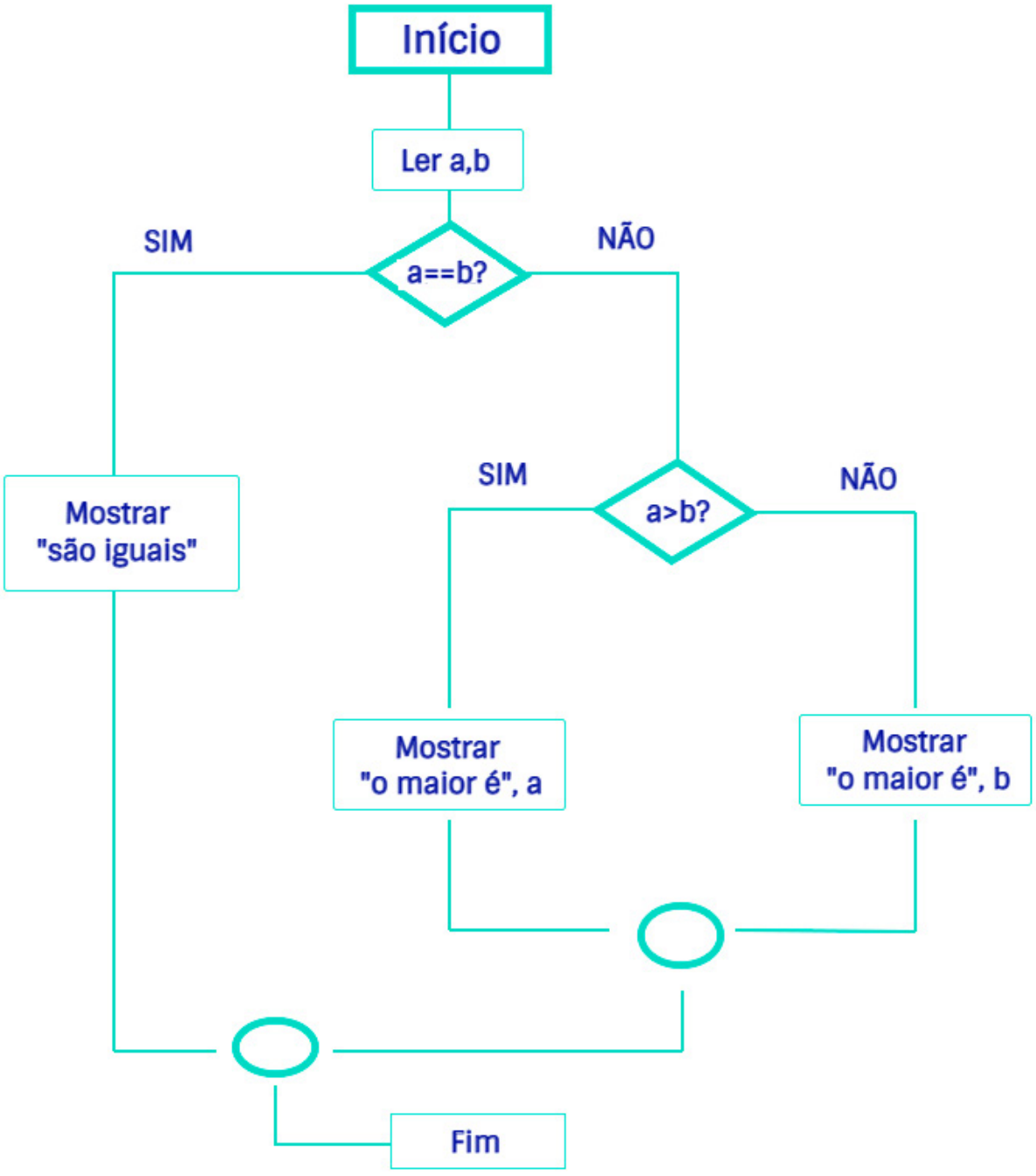
New Technology School

Tokio.

5.1 Condicionais em Python

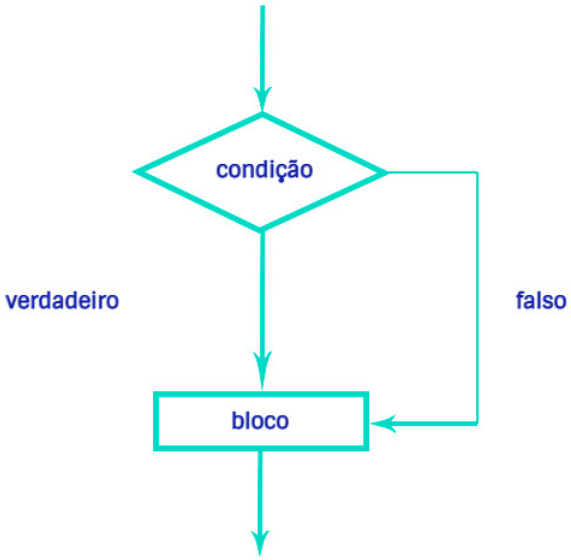
Uma expressão condicional, como o seu próprio nome indica, é uma condição para escolher entre uma opção e outra e, no processo mental, normalmente manifesta-se com um "Se"; por exemplo: se (vai chover), traz o guarda-chuva.

Através de uma expressão condicional, selecionamos um bloco de comandos que serão executados quando essa condição for cumprida (for verdadeira). Para entender um pouco melhor as instruções condicionais, vejamos um diagrama de fluxo de um programa, com este tipo de instruções:



IF

Através da expressão *if* avaliamos uma condição e executamos uma série de instruções, no caso de a condição avaliada ser verdadeira. Caso contrário, a sequência de instruções, do programa, é continuada normalmente, sem executar nenhuma das instruções indicada na expressão *if*. O seu diagrama de fluxo é o seguinte:



A instrução *if* apresenta a seguinte sintaxe:

if **condição**:
    instruções a executar

Uma expressão *if* permite dividir o fluxo de um programa em diferentes caminhos. As instruções incluídas dentro de *if* são executadas sempre que a expressão verificada seja *True*:

```
In [314]: if True:
              print("Se cumprir a condição")
              print("Imprime esta mensagem")
```

```
Se cumprir a condição
Imprime esta mensagem
```

```
In [315]: if False:
              print("Se cumprir a condição")
              print("Imprime esta mensagem")
          print("Este print não pertence ao IF")
```

```
Se cumprir a condição
Imprime esta mensagem
Este print não pertence ao IF
```

Podemos encadear diferentes *if*:

```
In [ ]: a = 2
        if a == 2:
            print("se a contém o valor 2")
        if a == 5:
            print("se a contém o valor 5")
```

Ou aninhar *if* dentro de outros *if*:

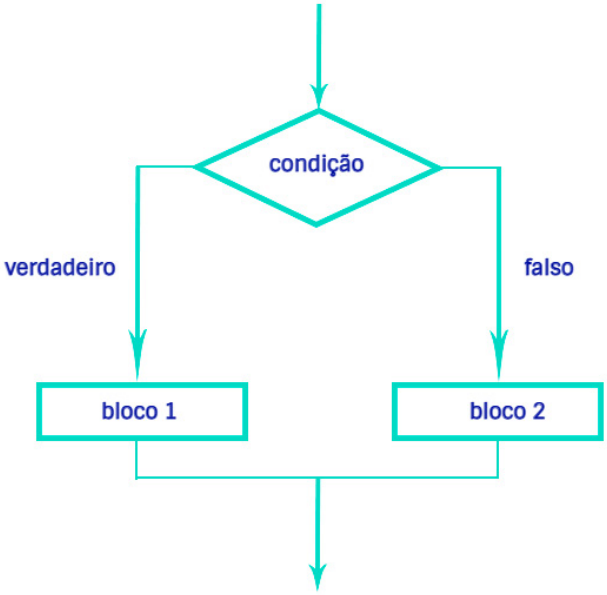
```
In [316]: a = 5
          b = 10
          if a == 5:
              print("se a contém o valor", a)
              b = b + 1
              if b == 10:
                  print("se b contém o valor", b)

          se a contém o valor 5
```

E como condição, podemos avaliar múltiplas expressões sempre que retorne *True* o *False*:

```
In [318]: if a == 5 and b == 11:
              print("se a contém o valor 5 e b contém o valor 10")

          se a contém o valor 5 e b contém o valor 10
```

IF _ ELSE:

Quando avaliamos uma condição, distinguimos o caso em que a **condição é verdadeira** (o *if* é cumprido, é verdadeiro), mas também podemos avaliar se a **condição é falsa** (*else* – se não). O diagrama de fluxo, neste caso, é o seguinte:



A instrução ***if ... else*** apresenta a seguinte sintaxe:

if **condição**:
    instruções a executar
else:
    instruções a executar

A expressão *else* é encadeada em *if* para verificar o caso contrário, que a condição não é cumprida:

```
In [320]: n = 6
          if n >= 0:
              print("é um número maior ou igual a 0")
          else:
              print("é um número menor que 0")
```

```
é um número maior ou igual a 0
```

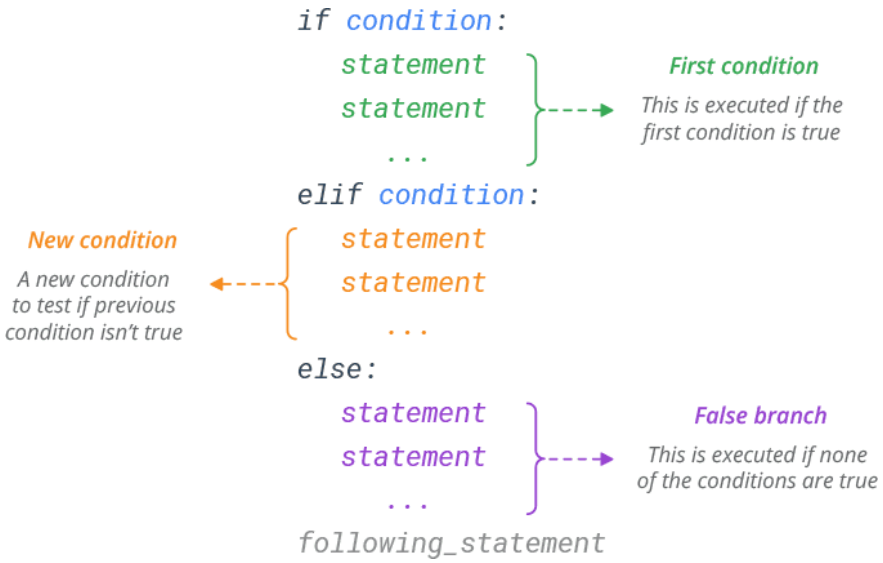
```
In [321]: n = 11
          if n % 2 == 0:
              print(n, "é um número par")
          else:
              print(n, "é um número impar")
```

```
11 é um número impar
```

IF – ELIF – ELSE:

A expressão *elif* dentro de uma expressão *if* é o resultado de aninhar várias expressões *if*, umas dentro das outras.

A forma de funcionamento é a seguinte: depois de avaliar a condição da expressão *if*, e se esta não for cumprida, realizamos outra avaliação com a expressão *elif* e no caso de não ser cumprida, continuaremos a avaliar as seguintes expressões *elif* até que alguma seja cumprida. Se nenhuma das condições anteriores for cumprida, executar-se-ão as instruções que vêm depois da expressão *else*. Se alguma das condições for cumprida, executar-se-á o código associado à mesma.



A instrução *if ... elif ... else* apresenta a seguinte sintaxe:

if **condição**:
    instruções a executar
elif **condição**:
    instruções a executar
else:
    instruções a executar

A instrução *elif* é encadeada em *if* ou outro *elif* para verificar múltiplas condições, sempre que as anteriores não sejam executadas.

In [323]:
```
comando = "OUTRA COISA"
if comando == "ENTRAR":
    print("Bem-vindo")
elif comando == "Saudar":
    print("Olá, espero que estejas bem a aprender Python")
elif comando == "Sair":
    print("A Sair do sistema...")
else:
    print("Este comando não é reconhecido")
```

```
Este comando não é reconhecido
```

In [327]:
```
nota = float(input("Introduza uma nota: "))
if nota >= 18:
    print("Excelente")
elif nota >= 16 and nota < 18:
    print("Muito Bom")
elif nota >= 14 and nota < 16:
    print("Bom")
elif nota >= 12 and nota < 14:
    print("Satisfaz")
elif nota >= 10 and nota < 12:
    print("Suficiente")
else:
    print("Insuficiente")
```

```
Introduza uma nota: 9
Insuficiente
```

É possível simular o funcionamento de *elif* com *if*, utilizando expressões condicionais, mas isto faz com que sejam executadas todas as instruções, ainda que não seja necessário.

In [325]:
```
nota = float(input("Introduza uma nota: "))
if nota >= 18:
    print("Excelente")
if nota >= 16 and nota < 18:
    print("Muito Bom")
if nota >= 14 and nota < 16:
    print("Bom")
if nota >= 12 and nota < 14:
    print("Satisfaz")
if nota >= 10 and nota < 12:
    print("Suficiente")
if nota < 10:
    print("Insuficiente")
```

```
Introduza uma nota: 17
Muito Bom
```

A instrução condicional *switch*, existente noutras linguagens de programação, não existe em Python, ainda que possa ser simulada através da utilização de um dicionário e funções. O *switch* é uma estrutura condicional múltipla, que permite avaliar, de forma rápida, se um valor coincide com uma grande lista de possibilidades. Por exemplo, se a variável **dia** valer 0, escolher-se-á segunda-feira, se a variável **dia** valer 1, escolher-se-á terça-feira, etc.:

```
switch(dia):
 case 0: “segunda-feira”,
 case 1: “terça-feira”,
 case 2: “quarta-feira”,
 case 3: “quinta-feira”,
 case 4: “sexta-feira”,
 case 5: “sábado”,
 case 6: “domingo”,
 default: “desconhecido”
```

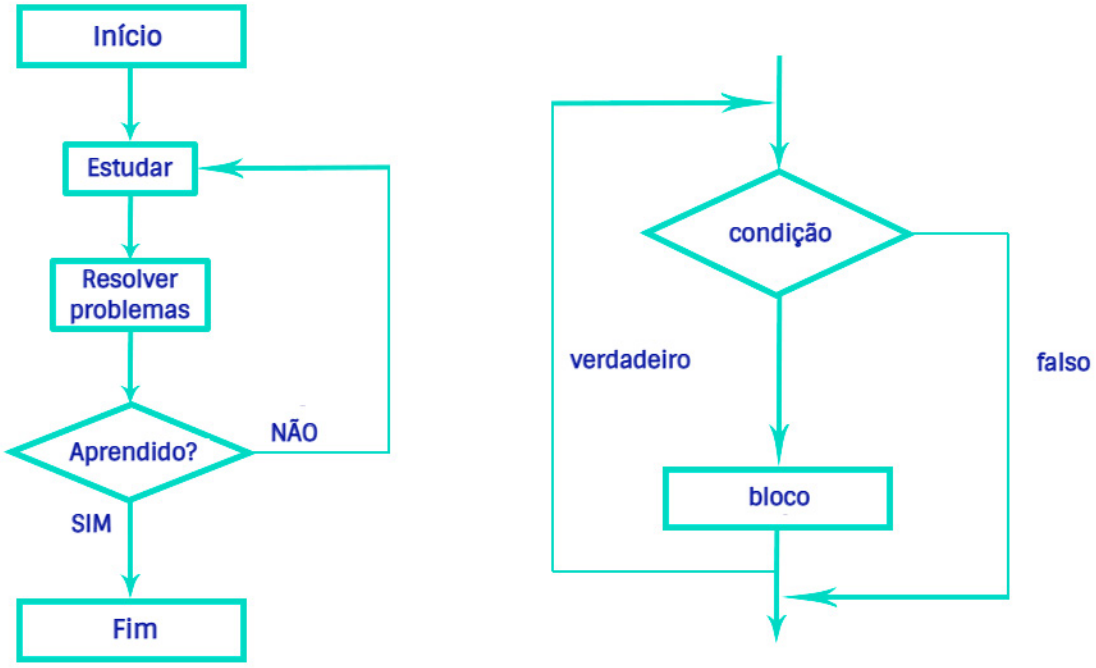
Devido ao facto de, em Python, ter sido eliminada a estrutura *switch*, veremos qual seria a alternativa para este exemplo:

In [329]:
```python
# Criamos um dicionário com as diferentes opções

dias = {
    0: "Segunda-feira",
    1: "Terça-feria",
    2: "Quarta-feira",
    3: "Quinta-feira",
    4: "Sexta-feira",
    5: "Sabado",
    6: "Domingo"
}

# Solicitamos ao utilizador um número de 0 a 6 que representa um dia da semana
optcaoUtilizador = int(input("Dias da semana que queremos mostrar (de 0 a 6): "))

# Com o número introduzido pelo usuário, acedemos ao dicionário e ao valor definido
nomedia = dias.get(optcaoUtilizador, "Desconhecido")

# Mostramos o resultado
print("O dia", optcaoUtilizador, "corresponde com: ", nomedia)
```

```
Dias da semana que queremos mostrar (de 0 a 6): 0
O dia 0 corresponde com:  Segunda-feira
```

5.2 *Loops* em Python

Um ***loop***, em programação, é uma sequência que executa, repetidas vezes, parte de um código, até que a condição atribuída a esse *loop* deixa de ser cumprida. O objetivo é repetir um bloco de código enquanto uma condição for verdadeira. Os *loops* mais utilizados, em programação, são o *loop while* e o *loop for*. Na imagem seguinte, podemos ver o diagrama de fluxo, de um programa, com um *loop* e o de um *loop* em si:

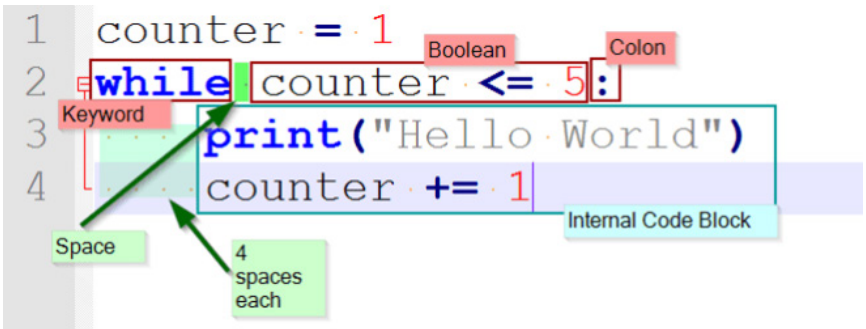


WHILE

Um *loop* com instrução *while* avalia uma condição antes de entrar no *loop* e, se for cumprida e enquanto for cumprida, executa as instruções que lhe estão associadas. A sintaxe é a seguinte:

While **condição**:
    instruções a executar

Um exemplo de um *loop while* é o seguinte:



O *loop do ... while*, existente noutras linguagens de programação, não existe em Python.

Um *loop while* baseia-se em iterar um bloco a partir da avaliação de uma condição lógica, sempre que esta seja *True*. Está nas mãos do programador decidir o momento em que a condição mudará para *False,* para fazer com que a instrução *while* seja finalizada.

"Iterar" é realizar uma ação várias vezes; cada vez que essa ação é repetida denomina-se de "iteração".

In [331]:
```
counter = 1
while counter <= 5:
    print("Hello world")
    counter += 1
```

```
Hello world
Hello world
Hello world
Hello world
Hello world
```

In [332]:
```
c = 0
while c <= 5:
    c +=1 # Equivale a c = c + 1
    print ("c tem o valor de ", c)
```

```
c tem o valor de  1
c tem o valor de  2
c tem o valor de  3
c tem o valor de  4
c tem o valor de  5
c tem o valor de  6
```

Uma expressão *else* pode ser encadeada num *loop while* para executar um bloco de código, uma vez que a condição já não retorna *True* (normalmente no final, ao sair do *loop*).

In [333]:
```
c = 0
while c <= 5:
    c += 1
    print("c tem o valor de", c)
else:
    print("Terminou a interação com o valor ",c)
```

```
c tem o valor de 1
c tem o valor de 2
c tem o valor de 3
c tem o valor de 4
c tem o valor de 5
c tem o valor de 6
Terminou a interação com o valor  6
```

Podemos "finalizar" a execução de *while* a qualquer momento, através da instrução *break*. Ao encontrar esta instrução, o programa sai imediatamente do *loop* em que se encontra, finalizando a iteração. Assim, o *else* não será executado, já que este só é chamado ao finalizar a iteração corretamente.

In [335]:
```
c = 0
while c <= 5:
    c += 1
    if (c==4):
        print("Terminamos o ciclo quando o valor for ", c)
        break
    print("c tem o valor de ", c)
else:
    print("Terminou a interação com o valor ", c)

print("programa continua...")
```

```
c tem o valor de  1
c tem o valor de  2
c tem o valor de  3
Terminamos o ciclo quando o valor for  4
programa continua...
```

Devemos ter especial atenção ao utilizar a instrução *break*, já que, com ela, estamos a finalizar o fluxo normal do programa.

Também podemos "saltar" a iteração atual, sem finalizar o *loop*, através da instrução *continue*:

In [336]:
```
c = 0
while c <= 5:
    c += 1
    if c == 3 or c == 4:
        # print("Continuamos com a seguinte interação", c)
        continue
    print("c tem o valor de ", c)
else:
    print("Terminou a interação com o valor de ", c)
```

```
c tem o valor de  1
c tem o valor de  2
c tem o valor de  5
c tem o valor de  6
Terminou a interação com o valor de  6
```

Com o *loop while* podemos criar um menu interativo:

In [338]:
```
print("Bem-vindo ao menu interativo")
while(True):
    print("""O que pretendes fazer? Escreve uma opção
    1) Saudar
    2) Somar dois números
    3) Sair
    """)
    option = input()
    if option == '1':
        print("Olá, espero que tenhas passado bem")
    elif option == '2':
        n1 = float(input("introduz o primeiro número: "))
        n2 = float(input("introduz o segundo número: "))
        print("O resultado da soma é: ", n1+n2)
    elif option == '3':
        print("Até logo! Foi um prazer ajudar-te")
        break
    else:
        print("Comando desconhecido. volta a tentar")
```

```
Bem-vindo ao menu interativo
O que pretendes fazer? Escreve uma opção
    1) Saudar
    2) Somar dois números
    3) Sair

1
Olá, espero que tenhas passado bem
O que pretendes fazer? Escreve uma opção
    1) Saudar
    2) Somar dois números
    3) Sair

2
introduz o primeiro número: 2
introduz o segundo número: 2
O resultado da soma é:  4.0
O que pretendes fazer? Escreve uma opção
    1) Saudar
    2) Somar dois números
    3) Sair

3
Até logo! Foi um prazer ajudar-te
```

Se a condição do *loop* for sempre cumprida, o *loop* não terminará nunca a sua execução e teremos o denominado *loop* infinito:

```
In [339]: contador = 0
          while True:
              print("Interacão", contador)
              contador += 1
```

```
Interacão 0
Interacão 1
Interacão 2
Interacão 3
Interacão 4
Interacão 5
Interacão 6
Interacão 7
Interacão 8
Interacão 9
Interacão 10
Interacão 11
Interacão 12
Interacão 13
Interacão 14
Interacão 15
Interacão 16
Interacão 17
Interacão 18
Interacão 19
```

À direita do botão "*Run*" temos o botão "*Stop*", com um quadrado no seu interior, para parar o processo, no caso de o nosso programa ficar dentro de um *loop* infinito.

Vejamos agora um exemplo prático de um *loop while*:

```
In [341]: n = 0
          while n < 10:
              if (n % 2) == 0:
                  print(n , 'é um número par')
              else:
                  print(n, "é um número impar")
              n = n + 1
```

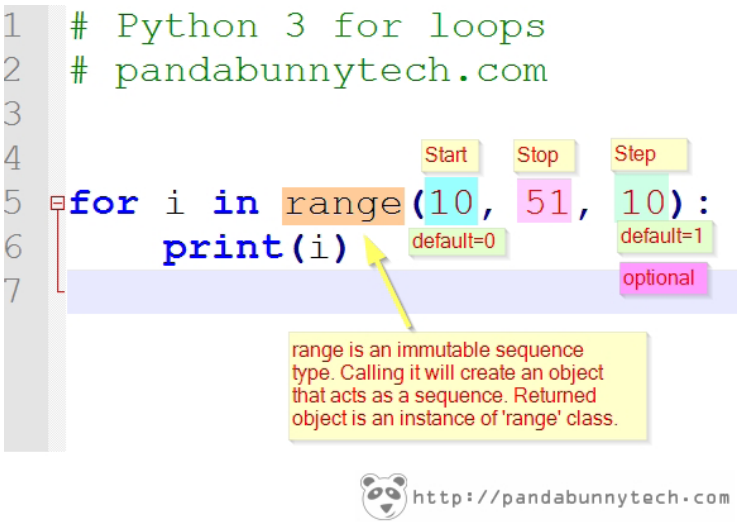
```
0 é um número par
1 é um número impar
2 é um número par
3 é um número impar
4 é um número par
5 é um número impar
6 é um número par
7 é um número impar
8 é um número par
9 é um número impar
```

FOR

Os *loops for* são outra forma de executar código, de modo repetitivo. Em vez de iterar sobre uma progressão aritmética de números, como é o caso de Pascal, ou oferecer ao utilizador a possibilidade de definir tanto o passo da iteração como a condição de fim, como é o caso de C, a expressão *for,* de Python, itera sobre os elementos de qualquer sequência, intervalo, lista ou cadeia de caracteres, seguindo a ordem que surge na sequência, intervalo, lista ou cadeia.

A instrução *for* apresenta a seguinte sintaxe:

for **variável** *in* **intervalo** ou **lista**/**tupla**/**cadeia**:
    instruções a executar



Para recordar, vamos recorrer aos elementos de uma lista, utilizando *while*:

```
In [ ]: numeros = [1,2,3,4,5,6,7,8,9,10]
indice = 0
while indice < len(numeros):
    print(numeros[indice])
    indice+=1
```

A função *range()* serve para gerar uma lista de números a que podemos recorrer facilmente, mas não ocupa memória, porque é interpretada sobre o funcionamento. Desta forma podemos simular um *loop for*, como é conhecido em qualquer outra linguagem de programação, por exemplo C:

```
In [15]: for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
In [ ]: range(10)
```

```
In [ ]: for i in [0,1,2,3,4,5,6,7,9]:
    print(i)
```

Se quisermos uma lista literal, podemos transformar o *range()* numa lista:

```
In [ ]: list(range(10))
```

Mas como podemos utilizar listas na expressão *for*? O modo mais fácil de recorrer a uma lista é da forma que vemos aqui:

```
In [342]: numeros = [1,2,3,4,5,6,7,8,9,10]
          for num in numeros:  # Para [variáveis] em [lista]
              print(num)
```

```
1
2
3
4
5
6
7
8
9
10
```

Desta forma, podemos modificar itens da lista. Para atribuir um novo valor aos elementos de uma lista, enquanto a examinamos, poderemos tentar atribuir ao número o novo valor:

```
In [ ]: print("Lista inicial: ")
        print(numeros)

        for numero in numeros:
            numero *= 10
            print(numero, end=", ")
```

Contudo, isto não funciona. A forma correta é fazendo referência ao índice da lista, em vez da variável:

```
In [ ]: indice = 0
        numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        for numero in numeros:
            numeros[indice] *= 10
            indice+=1
        numeros
```

Podemos utilizar a função *enumerate()* para conseguir, facilmente, o índice e o valor em cada iteração:

```
In [ ]: numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        for indice,numero in enumerate(numeros):
            numeros[indice] *= 10
        numeros
```

A expressão *for* pode ser utilizada com cadeias de caracteres:

```
In [343]: cadeia = "Olá amigos"
          for caracter in cadeia:
              print(caracter)
```

```
O
l
á

a
m
i
g
o
s
```

Contudo, recordemos que as cadeias são imutáveis, pelo que se tentarmos modificar o seu conteúdo, será produzido um erro.

```
In [344]: for i in cadeia:
              cadeia[i] = "'"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-344-1e1925eba876> in <module>
      1 for i in cadeia:
----> 2     cadeia[i] = "'"

TypeError: 'str' object does not support item assignment
```

Para resolver este inconveniente, podemos gerar uma nova cadeia:

```
In [345]: cadeia2 = ""
          for caracter in cadeia:
              if (caracter == " ") or (caracter == "H"):
                  cadeia += ""
              else:
                  cadeia2 += caracter

          print(cadeia)
          print(cadeia2)
```

```
Olá amigos
Oláamigos
```

```
In [346]: cadeia
Out[346]: 'Olá amigos'

In [347]: cadeia2
Out[347]: 'Oláamigos'
```

E, por último, recordemos que, se a condição do *loop* for sempre cumprida, o *loop* não terminará nunca a sua execução, e teremos o denominado *loop* infinito.

O *loop for* não inclui uma forma de implementar um *loop* infinito, tendo que realizar-se através de uma biblioteca, que veremos posteriormente.

```
In [349]: from itertools import count

          contador = 0
          for i in count():
              print("Interação: ", contador)
              contador += 1
```

```
Interação:  0
Interação:  1
Interação:  2
Interação:  3
Interação:  4
Interação:  5
Interação:  6
Interação:  7
Interação:  8
Interação:  9
Interação:  10
Interação:  11
Interação:  12
Interação:  13
Interação:  14
Interação:  15
Interação:  16
Interação:  17
Interação:  18
Interação:  19
```

Vejamos agora um exemplo prático de um *loop for*:

In [354]:
```
idiomas = ["ES_Espanha", "IT_Italia", "EN_Reino_Unido", "PT_Brasil", "PT_Portugal"]
contES = 0
contOL = 0
contPT = 0
for pais in idiomas:
    # print (pais[:2] # Para debug)
    if (pais[:2] == "ES"):
        print("Pais que fala espanhol ", pais)
        contES += 1
    elif (pais[:2] == "PT"):
        print("Pais que fala portugues ", pais)
        print(pais)
        contPT += 1
    else:
        print("Pais que fala outra linguagem ", end=" ")
        print(pais)
        contOL += 1

print("O número de paises que falam espanhol são: ", contES)
print("O número de paises que falam portugues são: ", contPT)
print("O número de paises que falam outras linguas são: ", contOL)
```

```
Pais que fala espanhol  ES_Espanha
Pais que fala outra linguagem  IT_Italia
Pais que fala outra linguagem  EN_Reino_Unido
Pais que fala portugues  PT_Brasil
PT_Brasil
Pais que fala portugues  PT_Portugal
PT_Portugal
O número de paises que falam espanhol são:  1
O número de paises que falam portugues são:  2
O número de paises que falam outras linguas são:  2
```