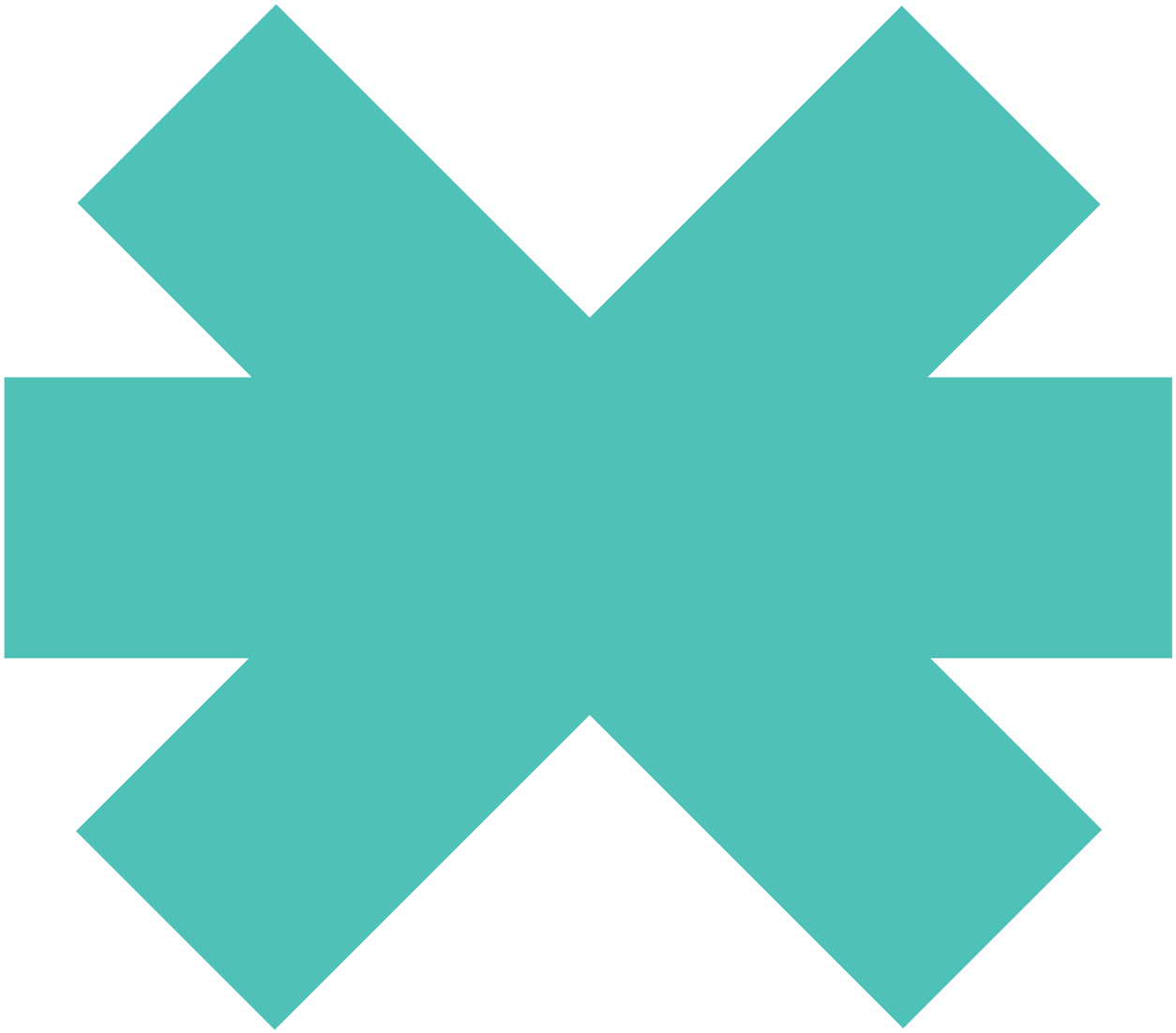
MÓDULO 2
Características básicas da linguagem
Programação Python

3

Entrada e saída de dados

New Technology School

Tokio.



3 Entrada e saída de dados

Sumário

New Technology School

Tokio.

3.1 Entrada de dados pelo teclado

A entrada de dados pelo teclado é realizada através da **função *input()*,** que nos retornará o valor teclado até premirmos a tecla *INTRO,* como uma cadeia de caracteres. Na imagem vemos a sua utilização e verificamos que o tipo de valor recebido é uma cadeia de caracteres:

```
In [271]: valor = input()
          Ola mundo

In [272]: valor
Out[272]: 'Ola mundo'

In [273]: print(type(valor))
          <class 'str'>


In [274]: valor = input()
          100

In [275]: # O valor do input dado pelo user é sempre obtido como uma String
          valor
Out[275]: '100'

In [276]: print(type(valor))
          <class 'str'>
```

Se à função *input()* passarmos como parâmetro uma cadeia de caracteres, essa cadeia surgir-nos-á como mensagem antes de ler o valor:

```
In [277]: # Podemos mostrar uma mensagem antes de ler um valor
          valor = input("Introduzir um valor: ")
          Introduzir um valor: 100
```

Ao receber um valor através desta função, há que ter em consideração que é sempre recebida uma cadeia de caracteres (*string*) e que não poderemos tratá-la como um número:

```
In [278]: # Não é possivel fazer operações com tipos diferentes
          valor + 100
          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-278-c28dd038cafe> in <module>
                1 # Não é possivel fazer operações com tipos diferentes
          ----> 2 valor + 100

          TypeError: can only concatenate str (not "int") to str
```

Para tratar o valor recebido, como um número inteiro, e poder operar com ele, devemos realizar um *casting* a um valor inteiro (*integer*) como se pode ver na imagem:

```
In [279]: valor = input("Introduzir um número inteiro: ")
          Introduzir um número inteiro: 500

In [280]: # A função int() faz a conversão, quando possivel, de um outro tipo para o tipo int
          valor = int(valor)

In [281]: valor
Out[281]: 500

In [282]: print(type(valor))
          <class 'int'>

In [283]: valor + 1000
Out[283]: 1500

In [284]: # Também se pode usar uma unica linha de código
          valor = int(input("Introduzir um numero inteiro: "))
          Introduzir um numero inteiro: 500

In [285]: print(type(valor))
          <class 'int'>
```

Se o que queremos é tratar o valor recebido como um número decimal e poder operar com ele, devemos realizar um *casting* a um valor decimal (*float*), como se vê na imagem:

```
In [287]: valor = input("Introduzir um número: ")
          Introduzir um número: 10.5

In [288]: # A função float() devolve um número com casas decimais ("ponto flutuante")
          valor = float(valor)

In [289]: 10 + valor
Out[289]: 20.5

In [290]: print(type(valor))
          <class 'float'>

In [291]: valor = float(input("Introduzir um número decimal ou inteiro: "))
          Introduzir um número decimal ou inteiro: 3.14

In [292]: valor
Out[292]: 3.14

In [293]: print(type(valor))
          <class 'float'>
```

Para permitir a introdução de valores múltiplos recorremos à instrução *for*, que veremos posteriormente:

```
In [294]: valores = []
          print("Introduza 3 valores")
          for x in range(3):
              valores.append( input("Introduza um valor: "))
          print(valores)
          Introduza 3 valores
          Introduza um valor: 5
          Introduza um valor: 6
          Introduza um valor: 7
          ['5', '6', '7']
```

3.2 Saída de dados pelo ecrã

Já vimos que a função *print()* é a forma usual de mostrar informação pelo ecrã. Agora vamos aprofundá-la um pouco mais:

```
In [296]: v = "outro texto"
          n = 10
          print("Um texto", v,"e um número", n)

          Um texto outro texto e um número 10
```

Esta função aceita caracteres especiais como as tabulações (/t) ou as quebras de linha (/n):

```
In [297]: print("Um texto\tuma tabulação")

          Um texto        uma tabulação

In [298]: print("Um texto\numa nova linha")

          Um texto
          uma nova linha
```

Para evitar os caracteres especiais, devemos indicar que uma cadeia é crua (*raw*), colocando um r antes da cadeia de caracteres a mostrar, como apresentado seguidamente:

```
In [299]: print("C:\nome\directório")

          C:
          ome\directório

In [300]: print(r"C:\nome\directório") # r -> raw (sentido literal)

          C:\nome\directório
```

Podemos utilizar """ *(aspas triplas)* para cadeias multilinha:

```
In [302]: print("""Uma linha
          outra linha
          outra linha\tuma tabulação""")

          Uma linha
          outra linha
          outra linha     uma tabulação
```

Utilizando o parâmetro *sep*, podemos separar cada um dos caracteres da cadeia, com o carácter indicado, e com o parâmetro *end* podemos finalizar a cadeia com o carácter que indiquemos:

```
In [2]: print(1, 2, 3, 4)
        print(1, 2, 3, 4, sep='*')
        print(1, 2, 3, 4, sep='#', end='&')

        1 2 3 4
        1*2*3*4
        1#2#3#4&
```

Existe uma funcionalidade nas cadeias de texto que nos permite formatar informação comodamente, utilizando identificadores referenciados, para isso usamos o método *format()*:

```
In [303]: texto = "Outro texto"
          num = 10
          c = "Um texto '{}' e um número '{}'".format(texto,num)
          print(c)
```

```
Um texto 'Outro texto' e um número '10'
```

Mediante o método *format()* também podemos referenciar a partir da posição dos valores, utilizando índices:

```
In [304]: print( "Um texto '{1}' e um número '{0}'".format(texto,num))
```

```
Um texto '10' e um número 'Outro texto'
```

Ou podemos utilizar um identificador com uma chave e passá-lo para o método *format()*:

```
In [16]: print( "Um texto '{t}' e um número '{n}'".format(t=texto,n=num))
```

```
Um texto 'Outro texto' e um número '10'
```

```
In [17]: print("{t},{t},{t}".format(t=texto))
```

```
Outro texto,Outro texto,Outro texto
```

Também, podemos realizar um formato avançado, alinhado à direita, em 30 caracteres, da seguinte forma:

```
In [306]: print( "{:>30}".format("palavra"))
```

```
                          palavra
```

Ou alinhado à esquerda em 30 caracteres:

```
In [307]: print( "{:30}".format("palavra"))
```

```
palavra
```

Ou centrado em 30 caracteres:

```
In [308]: print( "{:^30}".format("palavra"))
```

```
           palavra
```

Para realizar um truncamento de 3 caracteres procederemos de forma seguinte:

```
In [309]: print( "{:.5}".format("palavra"))
```

```
palav
```

Combinando vários destes formatos, realizamos um alinhamento à direita em 30 caracteres, com truncamento de 3:

```
In [310]: print( "{:>30.3}".format("palavra"))
```

```
                               pal
```

E, por último, para formatar números inteiros, preenchidos com zeros, realizaremos o seguinte:

```
In [23]: print("{:04d}".format(10))
         print("{:04d}".format(100))
         print("{:04d}".format(1000))

         0010
         0100
         1000
```

Se o quisermos fazer com números decimais ou flutuantes, faremos como mostrado:

```
In [311]: # Impressão de 7 caractéres (incluíndo o ponto décimal). Dos 7 caractéres, 4 serão décimais.
          print("{:07.3f}".format(3.1415926))
          print("{:07.3f}".format(153.21))

          003.142
          153.210
```