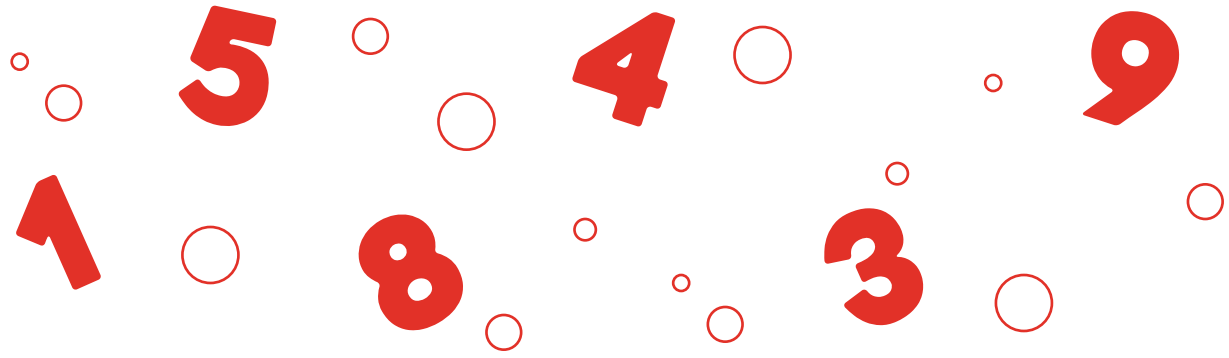
FUNDAMENTOS DA LINGUAGEM DE PROGRAMAÇÃO PYTHON

INTEIRO

Existem dois tipos básicos de variáveis do tipo número, um deles é o tipo INTEIRO



In [12]:

```
idade = 15
```

Para fazer cálculos entre variáveis e números, podem ser utilizados os operadores de **soma, subtração, multiplicação e divisão** normalmente

In [13]:

```
idade + 2
```

Out[13]:

17

In [16]:

```
numero = 2
```

In [18]:

```
idade * numero
```

Out[18]:

30

Observe que ao realizar a operação de **divisão**, por mais que resultasse em um **valor inteiro, o número veio como se fosse decimal**

In [32]:

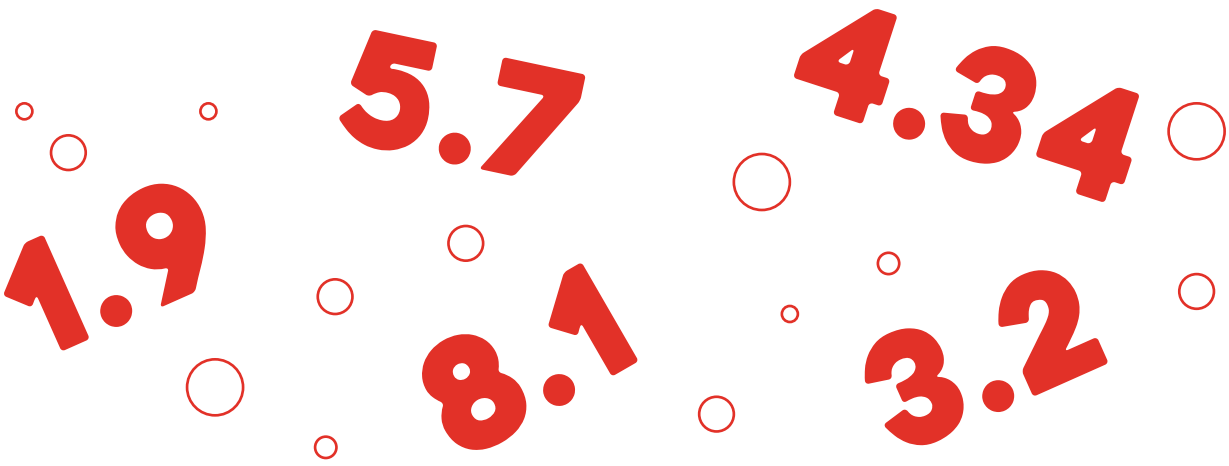
```
idade / 3
```

Out[32]:

```
5.0
```

FLOAT

Os **números decimais** em programação são chamados de **ponto flutuante** e são do tipo **FLOAT**



In [25]:

```
altura = 1.82
```

In [26]:

```
altura
```

Out[26]:

```
1.82
```

Como visto anteriormente, qualquer divisão entre **números inteiros** resulta em um **número decimal**

In [31]:

```
idade / 3
```

Out[31]:

```
5.0
```

Variáveis são muito utilizadas para **diminuir retrabalho** de algo que vai ser realizado **repetidas vezes**

In [30]:

```
ponto_flutuante = idade / 3
```

Além das operações aritméticas básicas, é possível realizar a **operação de potencialização** por meio de uma sequência de **dois asteriscos ****

In [44]:

```
2 ** 3
```

Out[44]:

8

Assim como na matemática, as expressões numéricas podem ser melhor formuladas por meio de **parêntesis**

In [46]:

```
(ponto_flutuante * 5) ** (1/2)
```

Out[46]:

5.0

LISTAS

É possível criar uma variável contendo um conjunto de tipos (diferentes ou não) com uma **LISTA**

In [13]:

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

Para acessar um valor específico da lista use o nome da variável e colchetes

In [14]:

```
lista[3]
```

Out[14]:

```
4
```

Esse acesso também funciona para alterar valores por meio de uma atribuição

In [15]:

```
lista[3] = 10
lista
```

Out[15]:

```
[1, 2, 3, 10, 5]
```

Também é possível fazer o acesso inverso utilizando números negativos

In [16]:

```
lista[-1]
```

Out[16]:

```
5
```

Além disso, existe uma operação chamada slicing , ou seja fatiamento, que fatia uma lista

In [20]:

```
lista
```

Out[20]:

```
[1, 2, 3, 10, 5]
```

In [21]:

```
lista[1:3]
```

Out[21]:

```
[2, 3]
```

In [22]:

```
lista[2:]
```

Out[22]:

```
[3, 10, 5]
```

In [23]:

```
lista[:2]
```

Out[23]:

```
[1, 2]
```

In [24]:

```
lista[:]
```

Out[24]:

```
[1, 2, 3, 10, 5]
```

STRING

Variáveis do tipo texto são chamadas de **STRINGS**, elas são identificadas por meio de aspas simples ou duplas.

XP Lab bemol

HELLO WORLD

Quando o tipo de dado é um texto, estamos falando das chamadas **strings**, elas são identificadas por meio de áspas simples ou duplas

In [25]:

```
nome = 'Giovana'
apelido = "Gi"
```

In [26]:

```
nome
```

Out[26]:

```
'Giovana'
```

In [27]:

```
apelido
```

Out[27]:

```
'Gi'
```

Para acessar uma letra específica use o nome da **variável e colchetes**

In [28]:

```
nome[0]
```

Out[28]:

```
'G'
```

Também é possível fazer o acesso inverso utilizando números negativos

In [29]:

```
nome[-3]
```

Out[29]:

```
'a'
```

Além disso, o **slicing** também funciona para **strings**

In [30]:

```
hashtag = '#estamosconectados'
```

In [31]:

```
hashtag[:8]
```

Out[31]:

```
'#estamos'
```

In [32]:

```
hashtag[8:]
```

Out[32]:

```
'conectados'
```

Ao colocar números entre áspas **simples ou duplas**, ele passa a ser uma string

In [33]:

```
idade = '15'
```

In [34]:

```
numero = '2'
```

A linguagem Python possui algumas operações que podem ser reallizadas com strings

In [35]:

```
nome.upper()
```

Out[35]:

```
'GIOVANA'
```

In [36]:

```
nome.lower()
```

Out[36]:

```
'giovana'
```

In [37]:

```
nome.title()
```

Out[37]:

```
'Giovana'
```

É possível **também concatenar (unir) duas variáveis do tipo string por meio do operador de soma +**

In [38]:

```
sobrenome = 'de Lucca'
```

In [39]:

```
nome + sobrenome
```

Out[39]:

```
'Giovanade Lucca'
```

In [40]:

```
nome + ' ' + sobrenome
```

Out[40]:

```
'Giovana de Lucca'
```

DICIONÁRIOS

Uma estrutura bem interessante da linguagem Python são os dicionários, os quais são inicializados por meio de chaves

In [113]:

```
dicionario = {}
```

Pode-se dizer que um dicionário é um lista que possui índices próprios

In [114]:

```
dicionario = {1: 'um', 2: 'dois'}
```

In [115]:

```
dicionario[1]
```

Out[115]:

```
'um'
```

Os índices de um dicionário são chamados de chaves e podem ser listados por meio da função **.keys()**

In [116]:

```
dicionario.keys()
```

Out[116]:

```
dict_keys([1, 2])
```

Os itens relativos a cada uma das chaves de um dicionário são chamados de valores e podem ser listados por meio da função **.values()**

In [117]:

```
dicionario.values()
```

Out[117]:

```
dict_values(['um', 'dois'])
```

É possível ter chaves e valores dos mais diversos tipos

In [118]:

```
d = {
    True: {'dois':2, 'cinco':5},
    False: [1,2,3]
}
```

BOOL

Em programação, existe um tipo diferente de variáveis chamado booleano do tipo **BOOL** que é uma representação binária, podendo ser apenas **verdadeiro (True) ou falso (False)**

In [40]:

```
verdadeiro = True
```

In [42]:

```
verdadeiro
```

Out[42]:

```
True
```

In [41]:

```
falso = False
```

In [43]:

```
type(falso)
```

Out[43]:

```
bool
```

ASSUNTOS COMPLEMENTARES

Para saber qual o tipo de uma determinada variável, utiliza-se a função **TYPE()**

```
In [24]:
type(idade)
Out[24]:
int

In [25]:
type(nome)
Out[25]:
str

In [26]:
type(nome[0])
Out[26]:
str

In [28]:
type(idade/3)
Out[28]:
float
```

Para criar relação entre variáveis e números, utilizam-se os chamados **operadores relacionais**

In [53]:

```
numero
```

Out[53]:

```
'2'
```

In [54]:

```
numero == 2
```

Out[54]:

```
False
```

In [55]:

```
type(numero)
```

Out[55]:

```
str
```

In [56]:

```
numero = 2
```

In [57]:

```
numero == 3
```

Out[57]:

In [58]:

```
numero != 3
```

Out[58]:

```
True
```

In [59]:

```
2 <= numero
```

Out[59]:

```
True
```

In [60]:

```
1 > numero
```

Out[60]:

```
False
```

Para mesclar os resultados de duas operações relacionais utilizam-se os **operadores lógicos and e or e not**

In [109]:

```
(1==1)and(2==3)
```

Out[109]:

```
False
```

In [110]:

```
(1==1)or(2==3)
```

Out[110]:

```
False
```

In [111]:

```
not (1==1)
```

Out[111]:

```
False
```

Para saber em qual caminho prosseguir baseado em uma determinada ação os algoritmos utilizam os **desvios condicionais**

In [61]:

```
if (3 == 3):
    print('Sim, 3 é igual a 3!')
```

```
Sim, 3 é igual a 3!
```

In [62]:

```
if (3 == 4):
    print('Sim, 3 é igual a 4!')
else:
    print('Não, 3 não é igual a 4!')
```

```
Não, 3 não é igual a 4!
```

In [63]:

```
if (3 == 5):
    print('Sim, 3 é igual a 5!')
elif (3 == 4):
    print('Sim, 3 é igual a 4!')
else:
    print('Não, 3 não é igual a 4 nem igual a 5!')
```

```
Não, 3 não é igual a 4 nem igual a 5!
```

É possível mesclar todos os assuntos aprendidos anteriormente e criar um desvio condicional com opedarores aritméticos e relacionais utilizando variváveis

In [64]:

```
variavel1 = '6'
variavel2 = '3'
```

In [65]:

```
if (variavel1 > variavel2):
    print(variavel1, 'é maior que', variavel2)
elif (variavel1 == variavel2):
    print(variavel1, 'é igual a', variavel2)
else:
    print(variavel1, 'é menor que', variavel2)
```

```
6 é maior que 3
```

INPUT()

Uma forma de se comunicar com o usuário é por meio da **FUNÇÃO INPUT()**

In [66]:

```
input('Qual seu nome? ')
```

```
Qual seu nome?
```

Out[66]:

```
''
```

In [74]:

```
nome = input('Qual seu nome? ')
```

```
Qual seu nome? Giovana
```

In [75]:

```
nome
```

Out[75]:

```
'Giovana'
```

In [78]:

```
numero = input('Escreve um número aqui: ')
```

```
Escreve um número aqui: 5
```

> O retorno da função input() gera um resultado do tipo string

In [79]:

```
numero * 3
```

Out[79]:

```
'555'
```

Para transformar esse resultado em tipo inteiro utilize a **função int()**

In [80]:

```
numero_int = int(numero)
numero_int * 3
```

Out[80]:

15

In [81]:

```
type(int(numero_int))
```

Out[81]:

int

Existem inúmeras formas de transformar tipos de variáveis

In [82]:

```
numero_str = str(numero_int)
numero_str
```

Out[82]:

'5'

In [83]:

```
list(numero_str*3)
```

Out[83]:

['5', '5', '5']

In [85]:

```
bool(numero_int)
```

Out[85]:

True

PRINT

Uma outra forma de se comunicar com o usuário, é por meio da função **PRINT**

In [86]:

```
print('Hello World!')
```

```
Hello World!
```

> Para juntar uma mensagem da função print com variáveis é só separar por vírgulas

In [89]:

```
print('Este é um número inteiro:', idade)
```

```
Este é um número inteiro: 15
```

Vamos praticar todos os assuntos aprendidos?

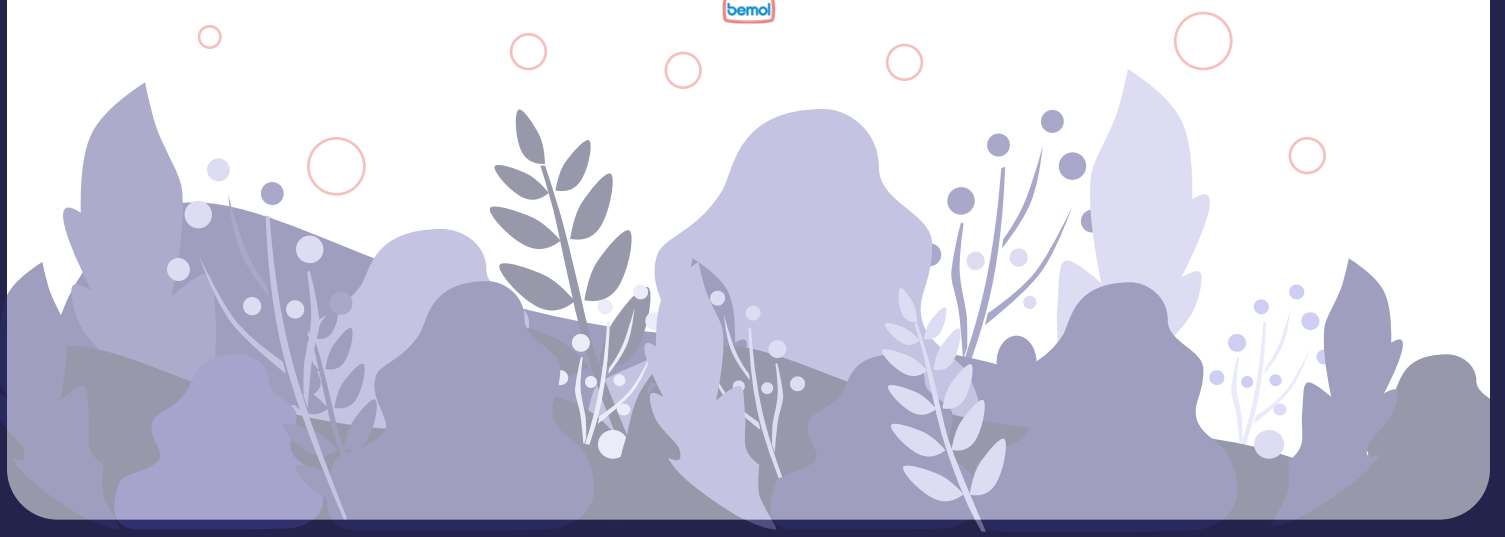
Fonte dos exercícios: Python Progressivo

1. Crie um programa que peça dois números e imprima o maior deles.

2. Crie um programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra imprimir:
   - F - Feminino;
   - M - Masculino;
   - Sexo Inválido.

3. Crie um programa que pede duas notas de um aluno. Em seguida ele deve calcular a média do aluno e dar o seguinte resultado:A mensagem "Aprovado", se a média alcançada for maior ou igual a sete;
   - A mensagem "Reprovado", se a média for menor do que sete;
   - A mensagem "Aprovado com Distinção", se a média for igual a dez.

XP Lab
bemol



FUNÇÕES

Quando existem atividades ou tarefas que sempre se repetirão podemos criar **FUNÇÕES** para fazê-la, evitando retrabalho

In [1]:

```
def nome_da_funcao (parametro):
    # atividades
    return parametro
```

Um exemplo de função sem parâmetros nem retorno é printar uma mensagem simples

In [2]:

```
def print_mensagem():
    print('Esta é uma função simples sem parâmetros nem retorno!')
```

In [3]:

```
print_mensagem()
```

Um exemplo de função sem parâmetros e com retorno é retornar a mensagem e printar após a chamada da função

In [4]:

```
def retorna_mensagem():
    return 'Esta é uma função sem parâmtros e com retorno!'

print(retorna_mensagem)
```

In [5]:

```
type(retorna_mensagem)
```

In [6]:

```
type(retorna_mensagem())
```

In [7]:

```
print(retorna_mensagem())
```

Normalmente as funções possuem **parâmetros e retorno**, mas isso não é uma regra

In [8]:

```
def par_ou_impar(numero):
    if (numero % 2 == 0):
        return 'Par'
    else:
        return 'Ímpar'
```

In [9]:

```
par_ou_impar(4)
```

In [10]:

```
par_ou_impar(3)
```

Também é possível **retornar mais de um valor em uma única função**

In [11]:

```
def calculadora_basica(num1, num2):
    return num1+num2, num1-num2, num1*num2, num1/num2
```

In [12]:

```
soma, subtracao, multiplicacao, divisao = calculadora_basica(6,2)
```

In [13]:

```
soma
```

In [14]:

```
subtracao
```

In [15]:

```
multiplicacao
```

In [16]:

```
divisao
```

Exemplos de funções prontas fornecidas pela linguagem Python

In [17]:

```
type(3)
```

In [18]:

```
round(12.34)
```

In [19]:

```
round(12.12345, 3)
```

In [20]:

```
int('15')
```

In [21]:

```
str(12)
```

Vamos praticar todos os assuntos aprendidos?

- 1. Faça uma função que receba dois números e retorne a média entre eles.

- 2. Faça uma função que leia um código de material que contenha 3 letras e quatro números (Exemplo: EAN1248) e retorne a parte texto (EAN) e número (1248) separadas.

- 3. Um vendedor de casquinhas de sorvete quer um sistema onde ele informe a quantidade, o sabor do sorvete e o total da compra. Em sua sorveteria vendem-se 3 sabores de sorvete com preços variados. Como podemos ajudá-lo com uma função para calcular o valor total do pedido para clientes que estão na fila?

    - Chocolate = 2.50
    - Baunilha = 3.00
    - Mista = 3.50

XP Lab
bemol