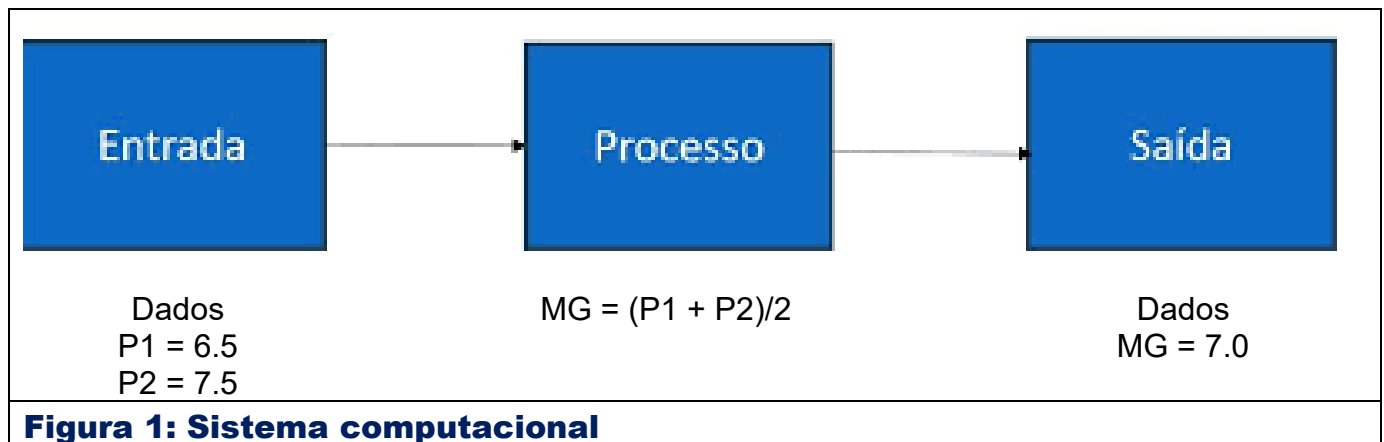


Aula 02 – Variáveis em Python 3

Parte 1 – Revisão Teórica

Partes de um sistema computacional



Pergunta:

Dados = Informação?

ENTRADA DE DADOS (Entrada Padrão de Dados)

Entrada de Dados refere-se a entrada de dados do mundo externo para o meio digital. É dessa forma que enviamos informações para dentro das nossas aplicações. A forma mais primitiva de enviar dados para uma aplicação é pelo Prompt de Comando, isto é, pelo Console propriamente dito.

```
python
C:\Program Files (x86)\Microsoft Visual Studio 14.0>python
Python 3.5.1 |Anaconda 4.0.0 (32-bit)| (default, Mar  4 2016, 15:28:01) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> input("informe a sua idade: ")
informe a sua idade:
```

Uma aplicação pode, a qualquer momento, pedir que o usuário informe uma determinada informação, para isto, basta invocar a função **input()**.

A partir da versão 3 do Python, a função **input()**, tem por objetivo de escrever a String passada como parâmetro e em seguida, ativar o modo de digitação, isto é, colocar o Console de uma determinada forma em que seja possível a digitação (entrada de dados)

A seguir, temos um exemplo onde imprimimos uma frase na tela e em seguida, é aberto para o usuário, a capacidade para digitação de letras e números.

```
input("Escreva entre aspas alguma informação a ser impressa no Console ")
```

O código acima, quando executado, irá imprimir o que está entre parêntesis no Console e em seguida, permitirá ao usuário a digitação de qualquer tipo de dado alfanumérico. O conjunto de caracteres digitado pelo usuário, será retornado pela função **input()**.

É responsabilidade do programador armazenar esses caracteres numa variável, como fazemos no exemplo a seguir:

```
num = input("Digite um número:")
print(num)
```

No exemplo acima, será impresso a frase definida entre aspas e então, o console entrará num modo que permitirá o usuário digitar texto e números. Após o usuário digitar o que desejar e pressionar a tecla Enter, os dados digitados serão retornados pela **função input()**.

No exemplo acima, estamos atribuindo o retorno da função para a variável num, logo, a mesma receberá aquilo que o usuário digitou no Console antes de pressionar a tecla Enter.

Por fim, imprimimos na tela os caracteres digitados pelo nosso usuário através da **função print()**.

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

Exemplos

```
nome = "Andres"
```

```
sobrenome = "Esteves"
```

print(nome,sobrenome)	print(nome,sobrenome, sep="-")	print(nome,sobrenome, sep=" ",
Resultado: Andres Esteves	Andres-Esteves	Andres Esteves

<pre>sujeito = "Python" verbo = "é" predicado= "fantástico" print(sujeito, verbo, predicado, sep="-", end="!\n")</pre>
Resultado: Python-é-fantástico!

OUTRAS FORMAS PARA ENTRADA DE DADOS

Uma aplicação pode receber dados através das mais distintas formas, como por exemplo,

- a) pela leitura de um **arquivo**,
- b) através dos **protocolos TCP/IP**,
- c) pelo uso de **janelas gráficas**, como por exemplo, a **biblioteca TkInter**, que é a forma nativa para a construção de Janelas Gráficas no Python e etc.

Há diversas maneiras para recebermos informações, porém, o princípio de como trataremos esses dados será, na maior parte da vezes o mesmo.

Certifique-se estar utilizando a versão 3.x do Python, do contrário, a função input() terá funcionamento anômalo e a sua aplicação não será executada da maneira correta! Por isso lembre-se: se você estiver tendo problemas, averigue a versão do Python instalada em seu computador!

EXEMPLO

```
login = input("Login:")
senha = input("Senha:")

print("O usuário informado foi: %s, e a senha digitada foi: %s" %(login, senha))
```

DEFINIÇÃO DE VARIÁVEL

O conceito de variável em programação é semelhante ao conceito de **variável na matemática**. As variáveis, de maneira geral, possuem um grupo de características que as definem e as tornam entidades únicas.

As características, ou então, as propriedades que todas as variáveis possuem, determinam um aspecto do funcionamento e também, especificam o seu ciclo de vida, a maneira como a informação são gerenciadas e manipuladas pela **Máquina Virtual do Python**.

Variável é um espaço de memória que reservamos para armazenar valores temporários que estão sendo processados ou manipulados. Toda **variável** possui um **tipo** e este será inferido conforme a informação inicial que atribuímos para a variável. Uma variável pode ter o seu valor alterado a qualquer momento, porém, não é possível alterar o seu tipo durante a execução.

Toda variável irá armazenar o valor na **memória RAM (Random Access Memory)** e a mesma estará disponível enquanto houver eletricidade, ou o computador não seja reiniciado. Todo armazenamento feito na memória RAM é considerado como um **armazenamento temporário**, e trabalhamos dessa maneira, devido a velocidade de entrega que memória RAM proporciona se comparada a outras mídias, como por exemplo, o **disco rígido, PenDrive, CD-Room** e etc.

Podemos declarar quantas variáveis desejarmos. O Python administrará (reservará espaço) para cada uma individualmente e associará o nome que definirmos ao espaço de memória. Temos então, que a localização do espaço de memória será feito por intermédio da referência (nome da variável) definida pelo programador. A partir do momento em que não fizermos mais uso de determinada variável, a máquina virtual do Python desalocará o espaço reservado, devolvendo-o ao **sistema operacional(SO)**.

A melhor maneira para manipularmos informações seja num programa, como também num Script, é através da utilização de variáveis. Isso porque, variáveis possuem uma estrutura simples e rápida para utilização.

COMO FUNCIONAM AS VARIÁVEIS?

Devemos pensar nas variáveis como sendo um espaço físico. Este espaço é administrado

- a) pelo hardware,
- b) pela placa-mãe,
- c) processador e
- d) sistema operacional (SO).

Cada pedaço do espaço físico possui um número que o **identifica (número identificador)** e assim, ao declararmos uma variável, estaremos, reservamos um **espaço físico** para guardar informações temporariamente e vinculando esse espaço ao nome da variável (referência).

As informações que serão armazenadas são temporárias, até porque, o funcionamento da memória RAM necessita, obrigatoriamente, de eletricidade para seu funcionamento. Não existe limites pré-definidos da quantidade de variáveis, ou quantidade de informações que é possível utilizar. Essas quantias sempre serão definidas pela quantidade de memória física que houver no sistema, e a quantidade de variáveis que podemos declarar, também estará, diretamente relacionada com a quantidade de espaço físico disponível, isto é, a quantidade de memória RAM existente.

CARACTERÍSTICAS DAS VARIÁVEIS

Há 4 características elementares referentes a todas as variáveis, são elas:

Ainda que não declaramos explicitamente o tipo das variáveis em Python, temos que uma variável inferida pela VM como sendo do tipo inteiro, não poderá, durante o tempo de execução, ter uma String ou então, qualquer outro valor atribuído. Por isso dizemos que o Python é uma linguagem fortemente tipada, isto é, as variáveis serão do mesmo tipo do inferido inicialmente.

Característica	Descrição
Nome (Referência)	<ul style="list-style-type: none"> ✓ Toda variável possui nome (a referência) é como iremos nos referir a um determinado espaço de memória durante o tempo de desenvolvimento. ✓ Toda vez que utilizarmos o nome de uma variável, poderemos ler o valor contido, ou então, atribuir quaisquer outros valores, desde que o tipo seja o mesmo. ✓ É importante entender que o nome, também chamado de referência, é uma ponte entre o endereço de memória e o nosso código. ✓ Assim, não precisamos manipular números (endereços de memória) durante o desenvolvimento, o que torna nossos códigos mais legíveis e fáceis de serem implementados.
Tipo	<ul style="list-style-type: none"> ✓ O Tipo de Variável funciona como uma espécie de classificação das informações, ou seja, o tipo informa com antecedência qual informação podemos esperar de determinada variável. ✓ Toda variável declarada será, obrigatoriamente, de algum tipo. ✓ O tipo da variável será armazenado junto com o valor que esta estará armazenando. <ul style="list-style-type: none"> a) Números b) Booleanos c) Strings d) Lista e) Tupla f) Dicionário
Tamanho (Espaço)	<ul style="list-style-type: none"> ✓ As informações precisam ser armazenada em alguma mídia, e as variáveis, são armazenadas na memória RAM do nosso computador. ✓ A memória RAM é formada por centenas de milhares de pequenos blocos, onde, cada bloco, possui um número que o identifica e o distingue dos demais. ✓ Assim, uma informação sempre estará armazenada numa determinada posição física de memória.
Valor	<ul style="list-style-type: none"> ✓ O valor, ou então, a informação contida na variável, isto é, o dado propriamente dito, é, a parte principal de qualquer variável. ✓ Toda variável será capaz de ter o seu valor alterado durante o tempo de execução, bastando para isso, somente, a atribuição de outro valor. A única regra definida para a atribuição de valores é que o tipo do valor a ser atribuído, seja do mesmo tipo da variável. ✓ Assim, para variáveis inteiras, só poderemos atribuir números.

- | | |
|--|---|
| | <ul style="list-style-type: none">✓ Para variáveis do tipo String, só poderemos atribuir texto.✓ Para variáveis do tipo Boolean, só podemos atribuir números entre 0 à 1, ou então, True ou False. |
|--|---|

1) Variável

Uma variável é uma área de memória onde armazenamos um valor qualquer. Esta área é associada a um nome ou identificador. Acessamos a área de memória através deste nome.

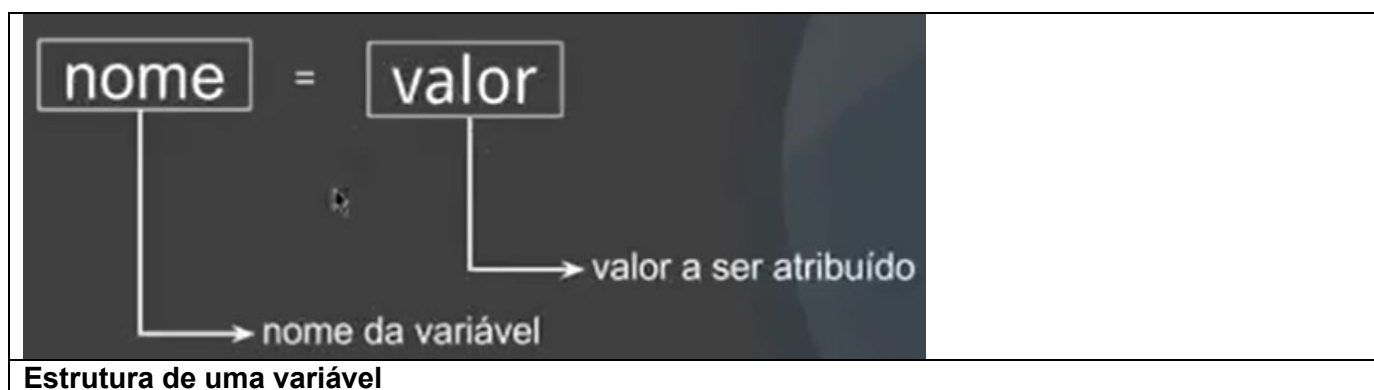
Python simplificou o tratamento das variáveis para o desenvolvedor.

Com o Python você não precisa **declarar nem o tipo nem o nome da variável antes de utilizá-la (tipagem dinâmica)**.

No Python isto ocorre automaticamente na primeira atribuição.

NOMENCLATURA

As variáveis possuem obrigatoriamente um nome que as identificam e as distinguem. Esse **nome (referência)** deve seguir algumas regras, para evitar, por exemplo, colisões de nomes e também, para não "confundir" o interpretador.



Estrutura de uma variável

NOME

Na construção de referências, pode-se utilizar, quaisquer letras, sejam elas **maiúsculas** ou **minúsculas**. Variáveis **não devem** utilizar **nomes de classes** ou **pacotes**, ou seja, não devemos declarar uma variável de nome **"str"**, ou então, **"int"**, até porque, esses são nomes de classes definidas por padrão pela linguagem Python.

Porém, a linguagem não proíbe-nos a utilização de referências que sejam iguais a nomes de classes, porém, se assim o fizermos, poderemos "bagunçar" o ambiente de execução da nossa aplicação.

De maneira geral, temos a disposição todo o alfabeto para definirmos o nome de variáveis. Também podemos, por exemplo, compor o nome da variável utilizando números, o que estudaremos nos tópicos a seguir.

Desde já é importante dizer que em nenhuma situação será permitida a utilização de caracteres especiais na composição de nomes, isto é, não podemos utilizar a lista de caracteres a seguir quando estivermos definindo o nome de variáveis:

CARACTERES NUMÉRICOS

Uma referência pode conter números, desde que este não seja o primeiro caractere. Assim, referências como por exemplo, **9num**, ou então, **1var, não são permitidos**, até porque, o interpretador, não conseguirá distinguir se estamos definindo uma referência ou então, estamos trabalhando com números.

Então, a utilização de números é permitida só e somente só após o primeiro caractere, como podemos ver nos exemplos a seguir:

```
num1 = 1
caractere = 'x'
flag = True
a2b3 = "qualquer texto... "
```

Listagem 1

PALAVRAS RESERVADAS

É importante observar que o Python possui um conjunto de palavras reservadas e essas, também não poderão ser utilizadas na composição de nomes.

Além dessa regra é importante também estar atento às palavras reservadas da linguagem (**Figura 2**), que não podem ser utilizadas para nomear variáveis.

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Figura 2. Palavras reservadas da linguagem Python

CARACTERES ESPECIAIS

Caracteres especiais, como por exemplo, os caracteres latinos, como o **c-cedilha ç**, o **til ~** e **etc**, não são permitidos na composição de nomes de variáveis. O único caractere tido como especial e que pode ser utilizado é o **underline _**.

A seguir, temos uma lista com os caracteres que não podem ser utilizados na composição de referências:

- **ç**
- **/**
- **=**
- **!**
- **@**
- **#**
- **\$**
- **%**
- **&**
- **/**
- **()**
- **[]**
- **^**
- **~**
- **'**

Novamente, é importante dizer que caracteres especiais não podem ser utilizados, seja qual for a posição em que estes aparecerem na definição da referência!

A seguir, temos mais um exemplo dos caracteres que **não podem** estar na primeira posição de um nome de variável

```
4num = 0
3 = "informe seu nome"
2d = 4
```

As referências utilizadas no código acima são todas inválidas, e a razão, deve-se ao fato das mesmas iniciarem com caracteres numéricos. Como já dito, o primeiro caractere deve, obrigatoriamente, ser uma letra.

Podemos concluir também, que uma variável **JAMAIS** será composta por somente números, haja vista que o primeiro caractere precisa ser uma letra.

Para definirmos o nome de variáveis, temos que iniciar utilizando um caractere que esteja no intervalo de **a** à **z** ou **A** à **Z**. A única exceção a esta regra é o caractere **underline** **_**.

A seguir, temos alguns nomes de variáveis **válidos**

```
vvar = 5
_vVar = ""
_____idade = 19
```

ÚNICA EXCEÇÃO

O único caractere especial, que pode ser utilizado na composição de nomes, e que também pode estar situando em qualquer posição, é o caractere **underline** **_**.

```
_minhaVar = 55
_____texto = ""
```

DICA

Na dúvida sobre quais caracteres são permitidos, utilize somente letras sem acentuação e sem espaços, até porque, todas as letras são permitidas.

Agora, se você quer decorar o que pode e o que não pode, pense da seguinte maneira:

- a) **números só se for o segundo caractere**
- b) **letras que não estejam no alfabeto Inglês não permitidos**
- c) **caracteres que não são letras e não são número não podem ser utilizados!**

1.1) Tipos de variáveis em Python

Como na maioria das linguagens temos o conceito de variáveis e tipos de dados e apesar da linguagem ser tipada dinamicamente, ela é **fortemente tipada** como veremos em breve.

Tipo	Seção	Exemplos
bool		True ou False
int	Números	3
float	Números	3.3
str	Strings	'João da Silva' ou "João da Silva"
list	Listas	[1, 2, 'ab']
dict	Dicionários	{'nome': 'João da Silva', 'idade': 21}
NoneType		None

Tabela 1. Tipos básicos de dados em Python

Não há declaração de variáveis estáticas em Python, como em linguagens como C, Java ou C#. Nessas linguagens, indicamos o tipo e o nome das variáveis e ela já passa a existir.

Por exemplo:

```
int idade;
```

Repare que só declaramos o tipo e o nome da variável, sem ter atribuído o valor.

Em Python, a variável só passa a existir quando atribuímos um valor, como no exemplo abaixo:

```
idade = 12
```

Isso faz todo sentido, já que não temos uma declaração explícita do tipo, como na linguagem Java ou C#. O interpretador do Python não tem como assumir um tipo.

1.a) Números

Na prática, todos os programas vão manipular textos ou números. Por isso, precisamos saber trabalhar com esses tipos fundamentais.

No universo Python, os termos que usaremos são números e strings. Manipulá-los é relativamente simples até para não programadores. Primeiro, vamos conhecer os conceitos básicos, para depois explorar alguns exemplos e explicar mais detalhes dos seus comportamentos.

Em programas de computador, quase sempre vamos manipular números. Um dos primeiros passos para aprender uma linguagem de programação é saber como operar seus números.

Cada linguagem disponibiliza, por padrão, alguns tipos de números embutidos que estão presentes em sua sintaxe; há outros em módulos podem ser importados e são manipulados por meio de objetos e APIs. Python 3 possui três tipos de números embutidos

Números	Descrição
int	<ul style="list-style-type: none"> ✓ inteiros com sinal. Podem ser de qualquer tamanho ✓ Os números sem ponto decimal serão considerados inteiros ✓ Inteiros são virtualmente ilimitados (http://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex) e podem crescer até o limite da memória. ✓ Números inteiros são escritos normalmente como em sua forma literal (por exemplo, 100) e também podem ser gerados usando a função int() como em int('1') , no qual geramos o número inteiro 1 a partir de uma string.
float	<ul style="list-style-type: none"> ✓ valores reais em ponto flutuante ✓ os números com ponto decimal serão considerados ponto flutuante. ✓ Em Python, no interpretador padrão CPython, os float são implementados usando o tipo double do C (http://en.wikipedia.org/wiki/IEEE_754-2008). ✓ Entretanto, em outros interpretadores – como Jython –, isso é feito por meio do tipo flutuante de precisão dupla da plataforma abaixo, no caso a JVM. ✓ Números de ponto flutuante têm o caractere . – como em 2.5 –, para separar as casas decimais. ✓ Também podem ser gerados pela função float() – como em float(1) ou float('2.5') –, e são aceitas na função float() as strings nan e inf com prefixos opcionais + e - – por exemplo, float('+nan') .
complex	<ul style="list-style-type: none"> ✓ Números complexos ✓ Os números complexos também estão embutidos na linguagem, mas sua aplicação não é tão comum no dia a dia. ✓ Eles têm a parte imaginária e a real, sendo que cada uma delas é um float . ✓ Os números complexos, por exemplo, dão resposta como a raiz quadrada de um número negativo e são usados em domínios como engenharia elétrica, estatística etc. ✓ Os números complexos têm o sufixo j ou J , como em 1+2J . ✓ Para acessar a parte real e a parte imaginária, use (1+2j).real ou (1+2j).imag .

Padrão *Snake_Case*

O Python utiliza por convenção o padrão **Snake_Case** para nomes de variáveis (ou identificadores). Um exemplo de variáveis em **Snake_Case** são:

```
idade_esposa = 20
perfil_vip = 'Flávio Almeida'
recibos_em_atraso = 30
```

Listagem a: Padrão Snake_Case

Padrão *CamelCase*

Em outras linguagens também se usa o padrão **CamelCase**. O mesmo exemplo com **CamelCase** (que não é o padrão do Python):

```
idadeEsposa = 20
perfilVip = 'Flávio Almeida'
recibosEmAtraso = 30
```

Listagem b: CamelCase

Vamos seguir o padrão do Python nesse curso, que é o ***Snake_Case!***

Python como calculadora (Operadores matemáticos)

A linguagem Python possui operadores que utilizam símbolos especiais para representar operações de cálculos, assim como na matemática:

Também conhecemos diversos operadores, como listados na tabela 2.

Simbolo	Descrição
+	Soma ou Concatenação
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão de inteiros
**	Exponenciação
%	Módulo da divisão de inteiros
=	Atribuição, coloca o resultado da expressão a direita na identificação (variável) a esquerda

Tabela 2. Operadores

Mesmo os tipos básicos em Python são implementados através de classes. E podem possuir métodos.

Devido ao suporte de sobrecarga de operadores, estes operadores podem ter funções diferentes dependendo das classes dos objetos na expressão.

Os números inteiros também podem ser expressos em diversas bases numéricas diferentes:

Operador	Descrição e exemplo
Soma (+)	<pre>>>> 2 + 3 5</pre> <p>Para utilizar números decimais, use o ponto no lugar de vírgula:</p> <pre>>>> 3.2 + 2.7 5.9</pre>
Subtração (-)	<pre>>>> 6 - 4 2</pre> <pre>>>> 7 - 8 -1</pre>

Multiplicação (*)	<pre>>>> 7 * 8 56 >>> 2 * 2 * 2 8</pre>
Divisão Real (/)	<p>Nesse caso, o resultado da divisão é real, mesmo que seja entre números inteiros.</p> <pre>>>> 100 / 20 5.0 >>> 10 / 3 3.3333333333333335</pre> <p>E se fizermos uma divisão por zero?</p> <pre>>>> 2 / 0</pre> <p>Traceback (most recent call last): File "<stdin>", line 1, in <module> ZeroDivisionError: division by zero</p> <p>Como não existe um resultado para a divisão pelo número zero, o Python interrompe a execução do programa (no caso a divisão) e mostra o erro que aconteceu, ou seja, «ZeroDivisionError: division by zero».</p>
Divisão inteira (//)	<p>O resultado da divisão é truncado para inteiro imediatamente inferior, mesmo quando aplicado em números reais, porém neste caso o resultado também será real.</p> <pre>>>> 10 // 3 3 >>> 666 // 137 4 >>> 666 / 137 4.861313868613139</pre>
Resto da divisão (%) Módulo (%).	<p>Retorna o resto da divisão.</p> <pre>>>> 10 % 2 0 >>> 10 % 3 1 >>> 666 % 137 118</pre>
Potência (**)	<p>Potenciação/Exponenciação: Pode ser utilizada para calcular a raiz através de expoentes fracionários. Por exemplo: 100*0.5.</p> <pre>>>> 2 ** 10 1024 >>> 10 ** 3 1000</pre>

```
>>> (10 ** 800 + 9 ** 1000) * 233
```

407254001651377825050774086265365912933271559572398924650169906751889900030955189004
9163474784706988806168855122018494451837288E a raiz quadrada?

Exemplos na parte II da aula 02

Funções para tipos numéricos

Além dos operadores, o programador também tem acesso a algumas funções para lidar com os tipos numéricos.

abs(): Retorna o valor absoluto do número.

oct(): Converte para octal.

hex(): Converte para hexadecimal.

pow(): Eleva um número por outro.

round(): Retorna um número real com o arredondamento especificado.

Base numérica	Exemplo	Descrição
Binário	0b111	7 em base decimal
Octal	0o1	8 em base decimal
Hexadecimal	0xff	255 em base decimal

Tabela 3. Literais para inteiros em outras bases numéricas

1.b) Variáveis do tipo Booleano

O tipo booleano em Python (bool) pode assumir um valor verdadeiro ou falso. O valor verdadeiro é chamado **True** e é igual a 1.

O valor falso é chamado **False** e é igual a zero.

Veja no exemplo a seguir

```
variavelBooleana = True # à variavel "variavel booleana é atribuido True
print ("Valor da variável de nome (variavelBooleana) = ",variavelBooleana, type(variavelBooleana));
exemplo26.py
```

Exemplos na parte II da aula 02

Temos alguns valores que, que são tratados como booleanos. Os mesmos são avaliados como falsos:

Valor	Descrição
False	Falso
None	Nulo
0	Zero
“ ” ou ‘ ‘	String Vazia
[]	Lista Vazia
()	Tupla Vazia
{}	Dicionário Vazio

Outras estruturas com o tamanho igual a zero também são tratados como **falso** e todos os outros tipos de objetos fora dessa lista são considerados verdadeiros.

O objeto **None**, que é **NoneType**, representa o **valor nulo** e é considerado **falso**.

1.c) Variáveis do tipo string

Uma variável string em Python é delimitada por <"> ou <'>.

Se você iniciou com <"> deve terminar com <">, e se iniciou com <'> deve terminar com <'>.

Você pode acessar um subconjunto da string através do operador [:]. Um único elemento da string pode ser acessado através do operador []. String podem ser concatenadas com o operador +. Para as situações nas quais a string deve ocupar mais de linha, utilizamos ""

Um exemplo de uso da variável string é apresentado a seguir:

FUNÇÃO TYPE()

A função **type(obj)**, com apenas um parâmetro, retorna o tipo (ou classe) de um objeto.

Na dúvida, use-a e descubra o tipo de um objeto referenciado por uma variável.

Para realizar testes em condicionais, é recomendado o uso da função `isinstance(obj, class)`.

Para meras verificações em explorações nos terminais, use `type()` quando tiver dúvida sobre o tipo de uma variável.

String é uma sequência

Em strings podemos acessar os elementos code points usando um índice e a notação **variavel[index]**. O índice varia de 0 até o tamanho da string menos 1.

Se ele for negativo, a contagem é na ordem inversa. A tabela 4 ilustra melhor:

P y t h o n						
+---+---+---+---+---+---+						
0	1	2	3	4	5	
-6	-5	-4	-3	-2	-1	

Tabela 4

Em Python, o termo sequência tem um significado especial. Para que um objeto seja uma sequência, como uma string, ele deve atender alguns requisitos. A seguir, veremos a aplicação de alguns conceitos de sequência em strings Python. Três das suas manipulações fundamentais são: saber tamanho, acessar um item por posição e acessar trechos por posições.

Para saber o tamanho, usamos **len(string)**.

Para acesso por índice, utilizamos **variavel[indice]**, ou para acesso de trechos, a **slice notation**, como em **minha_str[1:2]**. Exemplos na parte II da aula 02

Exemplos na parte II da aula 02

Por ser uma sequência, além do acesso por índices e slices, podemos executar outras operações como: `x in y`, se `x` está em `y`; `x not in y`, se `x` não está em `y`; `x + t`, concatenação de `x` com `y`; e `x * y`, `y` repetições de `x`.

```
"m" in "maracana"
```

```
True
```

```
"x" not in "maracana"
```

```
True
```

```
"m" + "aracana"
```

```
'maracana'
```

```
"a" * 3
```

```
'aaa'
```

Métodos no Strings

O tipo `str` serve para lidar com cadeias de texto, e entre os tipos básicos é um dos que mais métodos e recursos possui.

- ① **Strings** suportam indexação e fatiamento, nestes casos se comportam como uma lista de caracteres,
- ② O operador `in` avalia se a primeira *string* está contida na segunda, retornando um booleano
- ③ A **função `len()`** pode ser utilizada com qualquer objeto, e ela retorna o seu tamanho, a implementação específica depende de cada classe, no caso das *strings*, será o número de caracteres
- ④ O **método `lower()`** retorna uma nova *string* com todos os caracteres em minúsculo
- ⑤ O **método `upper()`** retorna uma nova *string* com todos os caracteres em maiúsculo
- ⑥ O **método `split()`** retorna uma nova lista de *strings*, cada elemento contendo uma palavra da *string* original

Existem muitos métodos disponíveis, além da possibilidade de uso da função `help()`, temos a documentação original com todas as opções: [String Methods](#).

Convertendo um tipo em outro

Uma vez instanciada uma variável, ela assume um determinado tipo. Mas podemos converter um tipo em outro. Veja os exemplos:

Inteiro para ponto flutuante	<pre>varInt= 50 # a variável varInt é um inteiro</pre> <pre>floatVar= float(varInt) # se imprimirmos floatVar teremos como resultado 50.0</pre>
Ponto flutuante para inteiro	<pre>varInt= int(varFloat)</pre> <p>Nesta conversão o valor após a virgula é desprezado. NÃO é arredondado.</p>
Número resultante de uma divisão de inteiros	<p>Quando você divide dois inteiros, o resultado pode ser automaticamente convertido em ponto flutuante, como no exemplo a seguir.</p> <pre>div = 7/2 = 3.5</pre>
Inteiro para string	<pre>varStr= str(varInteiro)</pre>

Número flutuante para string	<code>varStr= str(varNumeroFlutuante)</code>
String para inteiro	<code>varInt= str(varString)</code>
String para ponto flutuante	<code>varFloat= str(varString)</code>
Lista para tupla	<code>varList= [1,4,5,8,9]</code> <code>varTuple tuple(varList)</code>
Tuplas para lista	<code>varTupla = (1,4,7,9,10)</code> <code>varList= list(varTupla)</code>

Escopo das variáveis. Variáveis globais e locais.

Uma variável pode ser reconhecida por todas as funções ou métodos do programa, ou seja, pode ser uma **variável global**, ou reconhecida apenas dentro de um **bloco de código**, neste caso teremos uma **variável local**.

Variáveis globais são geralmente uma fonte de problemas, pois podem ser alteradas facilmente por muitos trechos do programa, o que dificulta a depuração. Porém as vezes permitem simplificar a implementação do programa. Basicamente, use com cuidado.

Toda variável instanciada dentro de um bloco de código é considerada local a este bloco. Variáveis instanciadas fora de qualquer bloco de código são variáveis globais.

Veja no exemplo a seguir: Atenção, os exemplos a seguir tem como objetivo demonstrar de maneira simples as diferenças entre os escopos local e global.

Existem formas mais “pitônicas” de se listar o conjunto de perguntas e respostas.

Função	Descrição
<code>int(x)</code>	Converte x em um número inteiro
<code>float(x)</code>	Converte x em um número fracionário
<code>str(x)</code>	Converte x em uma representação de string
<code>chr(x)</code>	Converte o inteiro x em um caractere
<code>ord(x)</code>	Converte o caractere x em seu valor inteiro
<code>oct(x)</code>	Converte o inteiro x em uma string octal
<code>hex(x)</code>	Converte o inteiro x em uma string hexadecimal

Figura 3: Conversão de dados

Glossário

atribuição (*assignment*)

Comando que atribui um valor a uma variável.

avaliar (*evaluate*)

Simplificar uma expressão através da realização de operações, para produzir um valor único.

comando (*statement*)

Trecho de código que representa uma instrução ou ação. Até agora, os comandos vistos foram de atribuição e exibição.

comentário (*comment*)

Informação em um programa dirigida a outros programadores (ou qualquer pessoa que esteja lendo o código fonte) e que não tem efeito na execução do programa.

composição (*composition*)

Habilidade de combinar expressões e comandos simples em expressões e comandos compostos, de forma a representar operações complexas de forma concisa.

concatenar (*concatenate*)

Juntar dois operandos lado a lado.

diagrama de estado (*state diagram*)

Representação gráfica de um conjunto de variáveis e os valores aos quais elas se referem.

divisão inteira (*integer division*)

Operação que divide um inteiro por outro e resulta em um inteiro. A divisão inteira resulta no número de vezes que o numerador é divisível pelo denominador e descarta qualquer resto.

expressão (*expression*)

Combinação de variáveis, operadores e valores, que representa um resultado único.

operando (*operand*)

Um dos valores sobre o qual o operador opera.

operador (*operator*)

Símbolo especial que representa uma computação simples, como adição, multiplicação ou concatenação de strings.

palavra-chave (*keyword*)

Palavra reservada usada pelo compilador/interpretador para analisar o programa; você não pode usar palavras-chave como `if`, `def`, e `while` como nomes de variáveis.

ponto-flutuante (*floating-point*)

Formato para representar números que possuem partes fracionárias.

regras de precedência (*rules of precedence*)

O conjunto de regras que governa a ordem em que expressões envolvendo múltiplos operadores e operandos são avaliadas.

tipo (*type*)

Um conjunto de valores. O tipo de um valor determina como ele pode ser usado em expressões. Até agora, os tipos vistos são: inteiros (tipo `int`), números em ponto-flutuante (tipo `float`) e strings (tipo `string`).

valor (*value*)

Um número ou string (ou outra coisa que ainda vamos conhecer) que pode ser atribuída a uma variável ou computada em uma expressão.

variável (*variable*)

Nome que se refere a um valor.

Bibliografia

Python, Escreva seus primeiros programas, Felipe Cruz, Casa doCodigo.

Curso Introdutório de Python, Grupy-Sanca, 09 de maio de 2019

Tutorial Python, Release 2.4.2. Guido van Rossum, Fred L. Drake, Jr., editor. Tradução: Python Brasil

Curso Python 3. Juracy Filho e Leonardo Leitão. Cod3r.com.br

https://aprendacompy.readthedocs.io/pt/latest/capitulo_01.html

<https://www.devmedia.com.br/python-trabalhando-com-variaveis/38644>

<https://cadernodelaboratorio.com.br/2017/12/01/variaveis-em-python3/>

Parte 2 – Prática (mão na massa)!!!

Para as atividades práticas vamos utilizar o VS Code, que deve estar previamente configurado para executar o python)

Para a criação dos arquivos (com extensão .py) vamos utilizar o editor de texto

Para executar, deve utilizar o interpretador do VS Code, temos 2 opções

- a) Apertar o Botão direito e selecionar a opção **RUN PYTHON TERMINAL**
- b) No menu superior, selecionar Debug -> **START WITHOUT DEBUGGING**

Para limpar o console, utilizar o comando clear ou cls

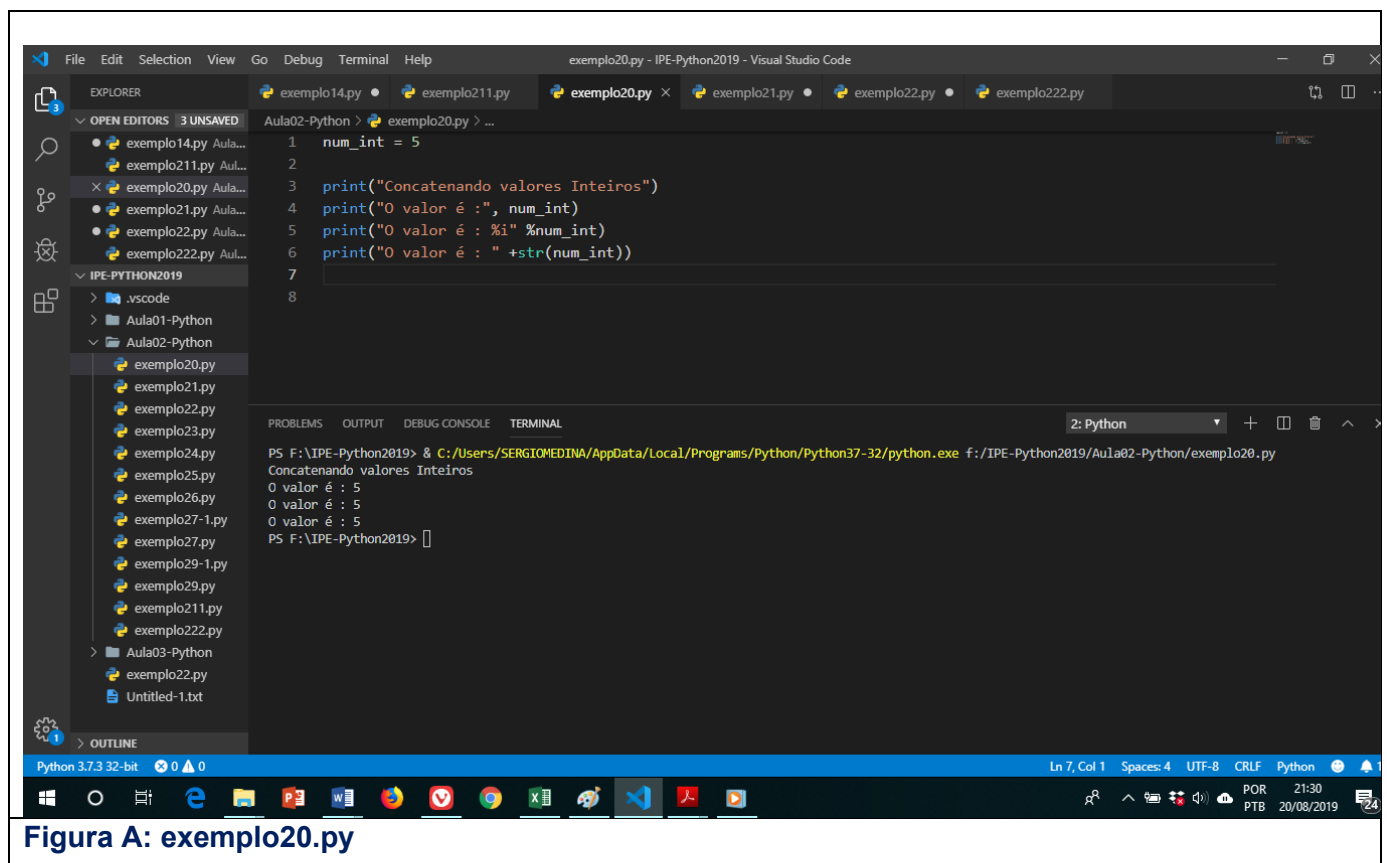
2) Criação do Projeto no VS CODE

- 1) Para isto crie uma pasta no seu pendrive com o nome **IPE-Python-2023**.
- 2) Dentro do **IPE-Python-2023**, criar a pasta **Aula02-Python**
- 3) Utilizando o VS CODE acesse a pasta criada (**IPE-Python-2023/ Aula02-Python**)
- 4) Para isso, acione a opção **File > New > File** e crie os seguintes (exemplo20.py, exemplo21.py, exemplo22.py, exemplo23.py e exemplo24.py), para melhor organização (**BOA PRATICA!!!!**)

Exemplo 20

PASTA: IPE-Python-2023/ Aula02-Python

arquivo: exemplo20.py



Exemplo 21

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo21.py

```

1  num_dec = 7.8
2
3  print("")
4  print("Concatenando valores decimais")
5  print("0 valor é :", num_dec)
6  print("0 valor é : %f" %num_dec + " com 5 casas decimais")
7  print("0 valor é : %.2f" %num_dec + " com 2 casas decimais")
8  print("0 valor é : %.10f" %num_dec + " com 10 casas decimais")
9  print("0 valor é : " +str(num_dec))
10
11

```

```

PS F:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe f:\IPE-Python2019\Aula02-Python\exemplo21.py
Concatenando valores decimais
0 valor é : 7.8
0 valor é : 7.800000 com 5 casas decimais
0 valor é : 7.80 com 2 casas decimais
0 valor é : 7.8000000000 com 10 casas decimais
0 valor é : 7.8
PS F:\IPE-Python2019>

```

Figura B: exemplo21.py

Exemplo 22

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo22.py

```

1  num_int = 5
2  num_dec = 7.8
3  var_texto = "Bom dia"
4
5
6  print("")
7  print("Concatenando valores decimais")
8  print("0 valor da string é :", var_texto)
9  print("0 valor da string é %s:" %var_texto)
10 print("0 valor da string é : " +var_texto)

```

```

PS F:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe f:\IPE-Python2019\Aula02-Python\exemplo22.py
Concatenando valores decimais
0 valor da string é : Bom dia
0 valor da string é Bom dia:
0 valor da string é :Bom dia
PS F:\IPE-Python2019>

```

Figura C: exemplo22.py

Exemplo 23

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo23.py

```

1 print("");
2 # Inteiro (int):
3 numero = 27;
4 print(numero);
5 print("Variavel numero = " + str(numero) + ", pertence ao tipo = ", type(numero));
6
7 # Real de ponto flutuante (float):
8 temperatura = 36.5;
9 print("");
10 print(temperatura);
11 print("Variavel temperatura = " + str(temperatura) + ", pertence ao tipo = ", type(temperatura));
12
13 # Complexo (complex):
14 c = 50 + 32j;
15 print("");
16 print(c);
17 print("Variavel Complexa = " + str(c) + ", pertence ao tipo = ", type(c));

```

Terminal output:

```

27
Variavel numero = 27, pertence ao tipo = <class 'int'>
36.5
Variavel temperatura = 36.5 , pertence ao tipo = <class 'float'>
(50+32j)
Variavel Complexa = (50+32j) , pertence ao tipo = <class 'complex'>
PS F:\IPE-Python2019>

```

Figura D: exemplo23.py

Exemplo 24

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo24.py

```

1 nome = "Andres Esteves";
2 idade = 22;
3 ativo = True;
4 altura = 1.76;
5
6 print(idade);
7 print(altura);
8 print(ativo);
9
10 print("A variavel nome = ", nome + ": pertence ao tipo = ", type (nome));
11 print("");
12 print("A variavel altura = ", str(altura) + ": pertence ao tipo", type (altura));
13 print("");
14 print("A variavel idade = ", str(idade) + ": pertence ao tipo", type (idade));
15 print("");
16 print("A variavel ativo = ", str(ativo) + ": pertence ao tipo", type (ativo));

```

Terminal output:

```

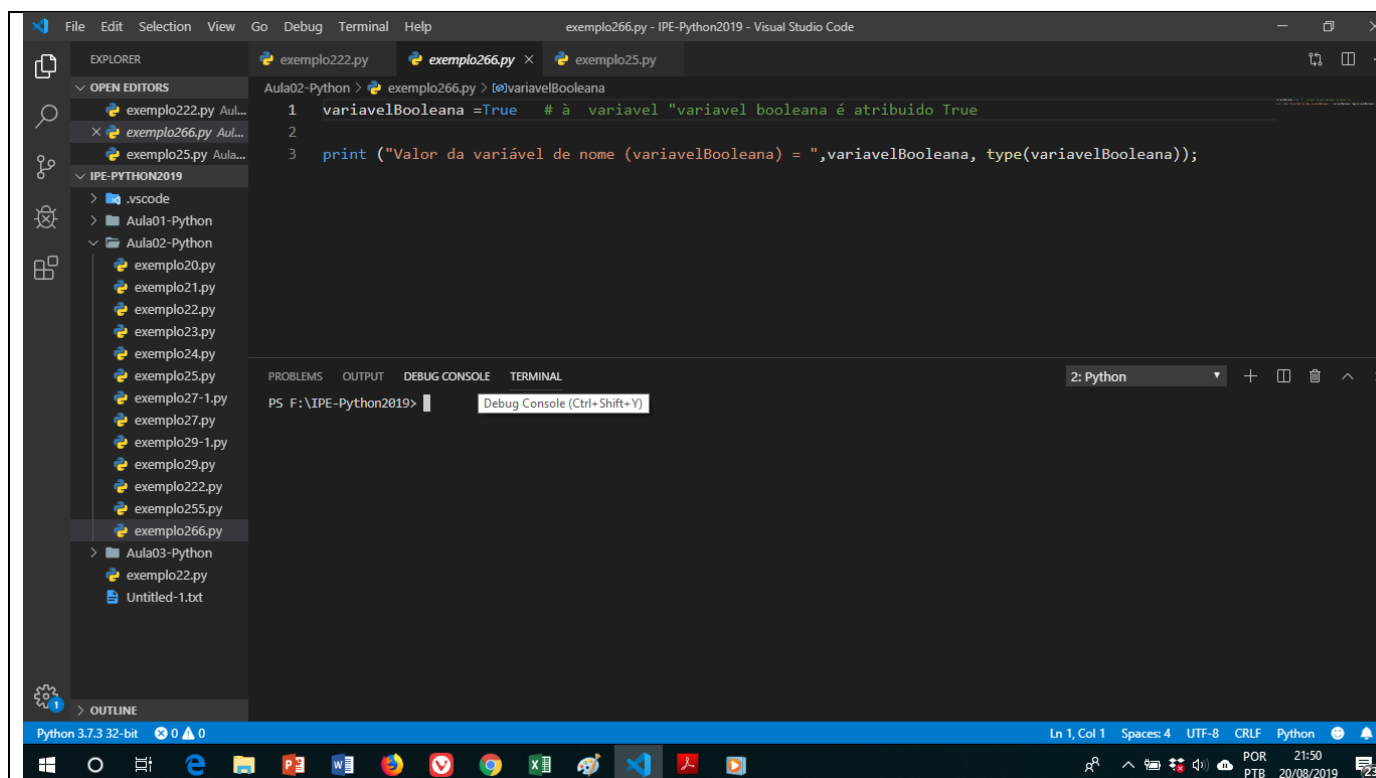
1.76
True
A variavel nome = Andres Esteves: pertence ao tipo = <class 'str'>
A variavel altura = 1.76: pertence ao tipo <class 'float'>
A variavel idade = 22: pertence ao tipo <class 'int'>
A variavel ativo = True: pertence ao tipo <class 'bool'>
PS F:\IPE-Python2019>

```

Figura E: exemplo24.py

Exemplo 25

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo25.py



```
File Edit Selection View Go Debug Terminal Help
exemplo266.py - IPE-Python2019 - Visual Studio Code

EXPLORER
OPEN EDITORS
exemplo222.py Aul...
exemplo266.py Aul...
exemplo25.py Aul...
IPE-PYTHON2019
.vscode
Aula01-Python
Aula02-Python
exemplo20.py
exemplo21.py
exemplo22.py
exemplo23.py
exemplo24.py
exemplo25.py
exemplo27-1.py
exemplo27.py
exemplo29-1.py
exemplo29.py
exemplo222.py
exemplo255.py
exemplo266.py
Aula03-Python
exemplo22.py
Untitled-1.txt

Aula02-Python > exemplo266.py > [te]variavelBooleana
1 variavelBooleana = True # à variavel "variavel booleana é atribuido True
2
3 print ("Valor da variável de nome (variavelBooleana) = ",variavelBooleana, type(variavelBooleana));

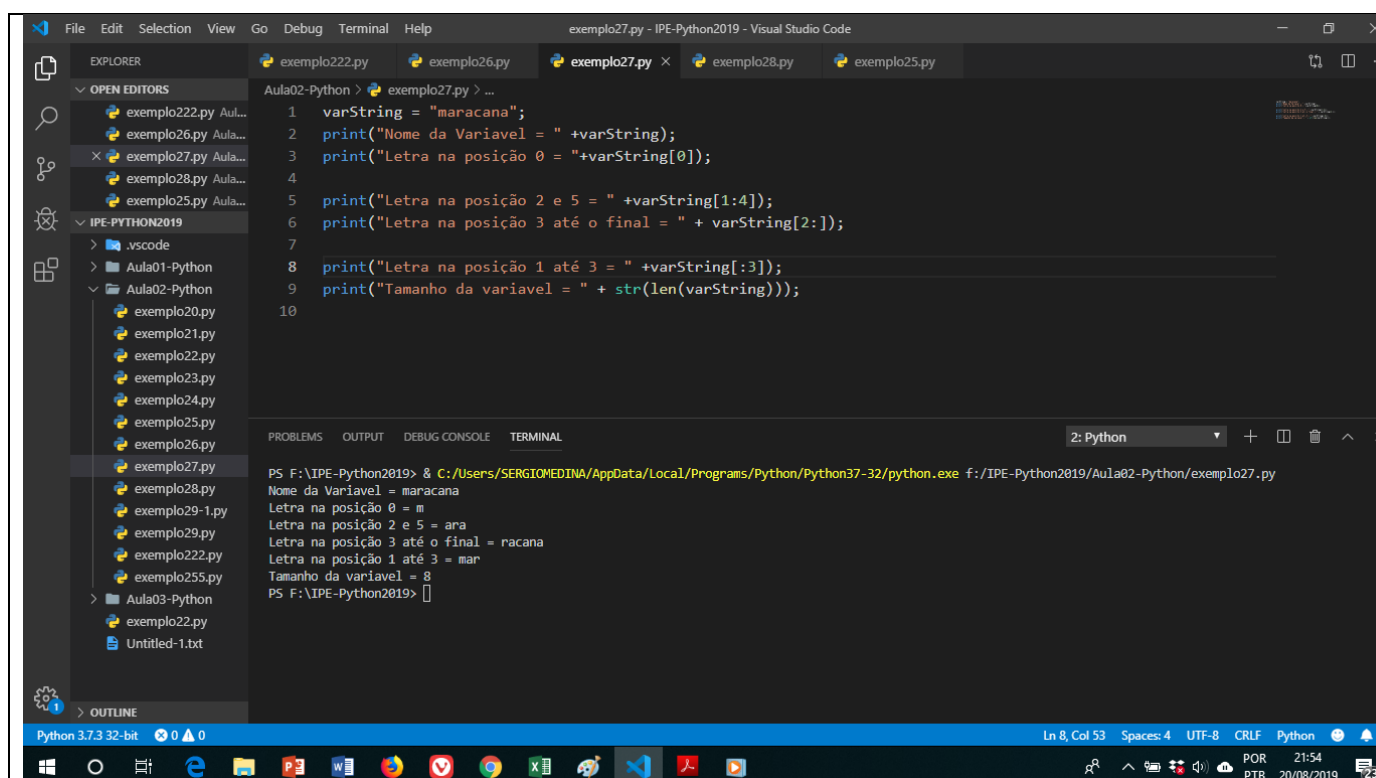
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS F:\IPE-Python2019> Debug Console (Ctrl+Shift+Y)

Python 3.7.3 32-bit 0 0 0 Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Python 21:50 20/08/2019
```

Figura F: exemplo25.py

Exemplo 26

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo26.py



```
File Edit Selection View Go Debug Terminal Help
exemplo27.py - IPE-Python2019 - Visual Studio Code

EXPLORER
OPEN EDITORS
exemplo222.py Aul...
exemplo26.py Aul...
exemplo27.py Aul...
exemplo28.py Aul...
exemplo25.py Aul...
IPE-PYTHON2019
.vscode
Aula01-Python
Aula02-Python
exemplo20.py
exemplo21.py
exemplo22.py
exemplo23.py
exemplo24.py
exemplo25.py
exemplo26.py
exemplo27.py
exemplo28.py
exemplo29-1.py
exemplo29.py
exemplo222.py
exemplo255.py
Aula03-Python
exemplo22.py
Untitled-1.txt

Aula02-Python > exemplo27.py > ...
1 varString = "maracana";
2 print("Nome da Variavel = " + varString);
3 print("Letra na posição 0 = " + varString[0]);
4
5 print("Letra na posição 2 e 5 = " + varString[1:4]);
6 print("Letra na posição 3 até o final = " + varString[2:]);
7
8 print("Letra na posição 1 até 3 = " + varString[:3]);
9 print("Tamanho da variavel = " + str(len(varString)));
10

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS F:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe f:/IPE-Python2019/Aula02-Python/exemplo27.py
Nome da Variavel = maracana
Letra na posição 0 = m
Letra na posição 2 e 5 = ara
Letra na posição 3 até o final = racana
Letra na posição 1 até 3 = mar
Tamanho da variavel = 8
PS F:\IPE-Python2019>

Python 3.7.3 32-bit 0 0 0 Ln 8, Col 53 Spaces: 4 UTF-8 CRLF Python 21:54 20/08/2019
```

Figura G: exemplo26.py

Exemplo 27

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo27.py

```

1  varString = "maracana";
2  print("Nome da Variavel = " + varString);
3  print("Letra na posição 0 = " + varString[0]);
4
5  print("Letra na posição 2 e 5 = " + varString[1:4]);
6  print("Letra na posição 3 até o final = " + varString[2:]);
7
8  print("Letra na posição 1 até 3 = " + varString[:3]);
9  print("Tamanho da variavel = " + str(len(varString)));
10

```

```

PS F:\IPE-Python2019> & C:/Users/SERGIONEDINA/AppData/Local/Programs/Python/Python37-32/python.exe f:/IPE-Python2019/Aula02-Python/exemplo27.py
Nome da Variavel = maracana
Letra na posição 0 = m
Letra na posição 2 e 5 = ara
Letra na posição 3 até o final = racana
Letra na posição 1 até 3 = mar
Tamanho da variavel = 8
PS F:\IPE-Python2019>

```

Figura H: exemplo27.py

Exemplo 28

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo28.py

```

1  numeroInteiro = 5 # a variável "numero inteiro" é do tipo inteiro
2
3  numeroInteiroBemGrande = 234987456123098
4  numeroPontoFlutuante = 34.874
5
6  print ("Valor da variável de nome (numeroInteiro) = ", numeroInteiro, type(numeroInteiro))
7
8  print("Valor da variável de nome (numeroInteiroBemGrande) = ", numeroInteiroBemGrande, type(numeroInteiroBemGrande))
9
10 print("Valor da variável de nome (numeroPontoFlutuante) = ", numeroPontoFlutuante, type(numeroPontoFlutuante))
11
12 # números complexos são comuns na área de ciências exatas. Se você não
13 # teve contato com números complexos, pode pular esta parte, pois
14 # certamente não precisará de utilizá-los
15
16 numeroComplexo= 5 + 4j # 4 é a parte imaginária do número
17 print("Valor da variável de nome (numeroComplexo) = ", numeroComplexo, type(numeroComplexo))

```

```

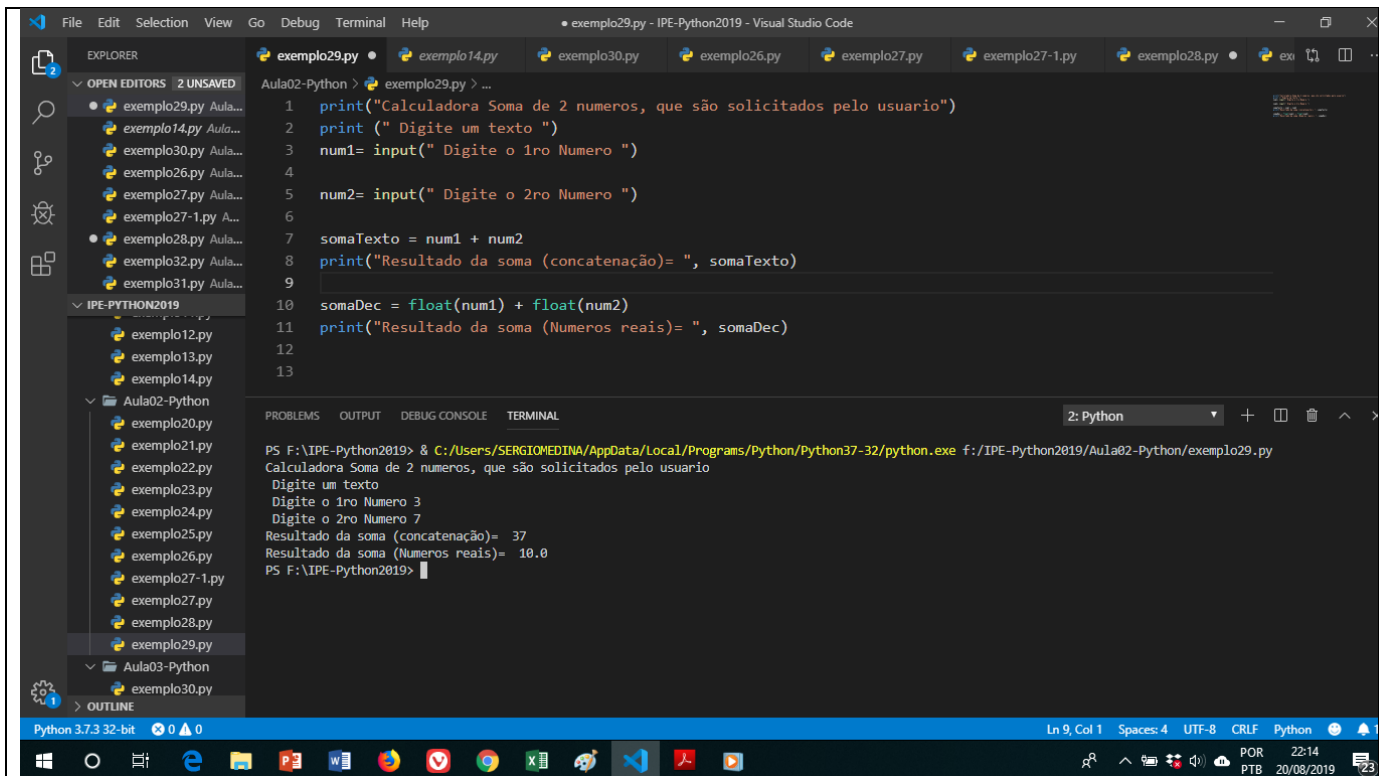
PS F:\IPE-Python2019> & C:/Users/SERGIONEDINA/AppData/Local/Programs/Python/Python37-32/python.exe f:/IPE-Python2019/Aula02-Python/exemplo28.py
Valor da variável de nome (numeroInteiro) = 5 <class 'int'>
Valor da variável de nome (numeroInteiroBemGrande) = 234987456123098 <class 'int'>
Valor da variável de nome (numeroPontoFlutuante) = 34.874 <class 'float'>
Valor da variável de nome (numeroComplexo) = (5+4j) <class 'complex'>
PS F:\IPE-Python2019>

```

Figura I: exemplo28.py

Exemplo 29

PASTA: IPE-Python-2023/ Aula02-Python
arquivo: exemplo29.py



```
File Edit Selection View Go Debug Terminal Help
exemplo29.py • exemplo14.py exemplo30.py exemplo26.py exemplo27.py exemplo27-1.py exemplo28.py exemplo29.py
OPEN EDITORS 2 UNSAVED
exemplo29.py Aula...
exemplo14.py Aula...
exemplo30.py Aula...
exemplo26.py Aula...
exemplo27.py Aula...
exemplo27-1.py A...
exemplo28.py Aula...
exemplo32.py Aula...
exemplo31.py Aula...
IPE-PYTHON2019
exemplo12.py
exemplo13.py
exemplo14.py
Aula02-Python
exemplo20.py
exemplo21.py
exemplo22.py
exemplo23.py
exemplo24.py
exemplo25.py
exemplo26.py
exemplo27-1.py
exemplo27.py
exemplo28.py
exemplo29.py
Aula03-Python
exemplo30.py
OUTLINE
Aula02-Python > exemplo29.py > ...
1 print("Calculadora Soma de 2 numeros, que são solicitados pelo usuario")
2 print(" Digite um texto ")
3 num1= input(" Digite o 1ro Numero ")
4
5 num2= input(" Digite o 2ro Numero ")
6
7 somaTexto = num1 + num2
8 print("Resultado da soma (concatenação)= ", somaTexto)
9
10 somaDec = float(num1) + float(num2)
11 print("Resultado da soma (Numeros reais)= ", somaDec)
12
13
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
PS F:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe f:\IPE-Python2019\Aula02-Python\exemplo29.py
Calculadora Soma de 2 numeros, que são solicitados pelo usuario
Digite um texto
Digite o 1ro Numero 3
Digite o 2ro Numero 7
Resultado da soma (concatenação)=  37
Resultado da soma (Numeros reais)=  10.0
PS F:\IPE-Python2019>
```

Figura J: exemplo29.py