Vitor Amadeu Souza

Machine Learning

programado em

Python

© 2024 by Cerne Tecnologia e Treinamento Ltda.

© 2024 by Vitor Amadeu Souza

Nenhuma parte desta publicação poderá ser reproduzida sem autorização prévia e escrita de **Cerne Tecnologia e Treinamento Ltda.** Este livro publica nomes comerciais e marcas registradas de produtos pertencentes a diversas companhias. O editor utiliza as marcas somente para fins editoriais e em benefício dos proprietários das marcas, sem nenhuma intenção de atingir seus direitos.

Dezembro de 2024

Direitos reservados por:
Cerne Tecnologia e Treinamento Ltda
Produção: Cerne Tecnologia e Treinamento
E-mail da Empresa: cerne@cerne-tec.com.br
Home Page: www.cerne-tec.com.br.com.br
Atendimento ao Consumidor: sac@cerne-tec.com.br
Contato com o Autor: vitor@cerne-tec.com.br

FEITO NO BRASIL

“Venha também sobre mim a tua benignidade, ó Senhor, e a tua salvação, segundo a tua palavra.”

Sl 119:41

Cerne Tecnologia

A Cerne Tecnologia tem uma equipe preparada para desenvolvimento de projetos eletrônicos em diversas áreas: Médica, Entretenimento, Industrial, Robótica, Científica, Automobilística, Aeronáutica, etc. Trabalhamos com tecnologia microcontrolada usando o PIC, ARM, AVR, 8051, dsPIC, PIC24, PIC32 além do Arduino, Raspberry, Beaglebone etc. Desenvolvemos o projeto desde sua concepção até a entrega do produto final, passando pelas etapas de esquema elétrico, protótipo e desenvolvimento de circuito impresso.

Desenvolvemos aplicativos para smartphones/tablets Android, iOS, Blackberry, Windows Phone e no desenvolvimento de softwares a nível PC para plataforma Windows, usando ferramentas como o Visual Basic, C# e C++.

Atuamos na parte de montagem de placas, onde podemos fornecer ambos os serviços de desenvolvimento de projetos e produção ou apenas um destes.

Desenvolvemos esquemas elétricos e layout de PCI, tanto em tecnologia convencional como SMD.

Temos a flexibilidade de customizar um de nossos produtos, de modo a atender a uma necessidade específica do cliente, tornando o custo de desenvolvimento menor se comparado a construção de um projeto desde a sua fase inicial.

Desenvolvemos e fornecemos kits didáticos para diversos microcontroladores além de apostilas, livros e e-books.

Na hora de desenvolver um projeto ou equipar seu laboratório não hesite em nos contatar. Entre em contato conosco através do endereço cerne-tec.com.br para obter mais informações.

Cerne
Conhecimento para o Desenvolvimento

cerne-tec.com.br

Sumário

Capítulo I
Metodologia de desenvolvimento

1. Introdução

Este livro oferece uma introdução abrangente à programação em Python e às técnicas de aprendizado de máquina, visando capacitar os leitores a extrair insights valiosos. Começando com uma metodologia de desenvolvimento sólida, o livro aborda conceitos fundamentais da programação, incluindo operadores, variáveis, tratamento de erros e manipulação de arquivos. À medida que avança, os capítulos exploram diversas técnicas de aprendizado de máquina, como a função sigmoide, perceptron e modelos de regressão linear, além de métodos de agrupamento, como Agglomerative Clustering e K-Means. Também são discutidos algoritmos de classificação, incluindo árvores de decisão, KNN e Naive Bayes.

O livro destaca a importância da visualização de dados e inclui tópicos avançados, como aprendizado profundo com LSTM, MLP e SVM, além de técnicas de redução de dimensionalidade, como PCA e t-SNE. Para interessados em processamento de linguagem natural, o capítulo sobre NLTK apresenta ferramentas para análise de texto. Com exemplos práticos e exercícios, esta obra é um guia teórico e prático que prepara os leitores para enfrentar desafios no campo da ciência de dados e aprendizado de máquina, fornecendo as ferramentas necessárias para se destacarem em suas jornadas profissionais e acadêmicas.

Capítulo II

Programação em Python

1. Introdução

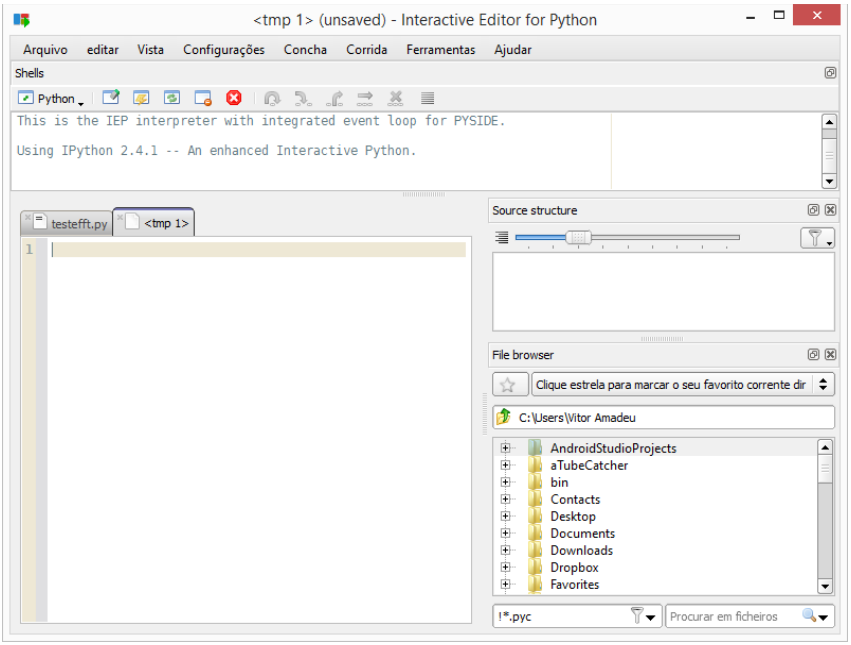
Nesta literatura a distribuição Pyzo foi utilizada, no qual a última versão pode ser baixada através do link abaixo.

http://www.pyzo.org/downloads.html

Baixe e instale também a última versão do Python disponível em:

https://www.python.org/

A vantagem desta distribuição é que ela por padrão já vem com as bibliotecas a serem utilizadas ao longo desta obra. Após a instalação inicialize o programa, a tela a seguir será apresentada.



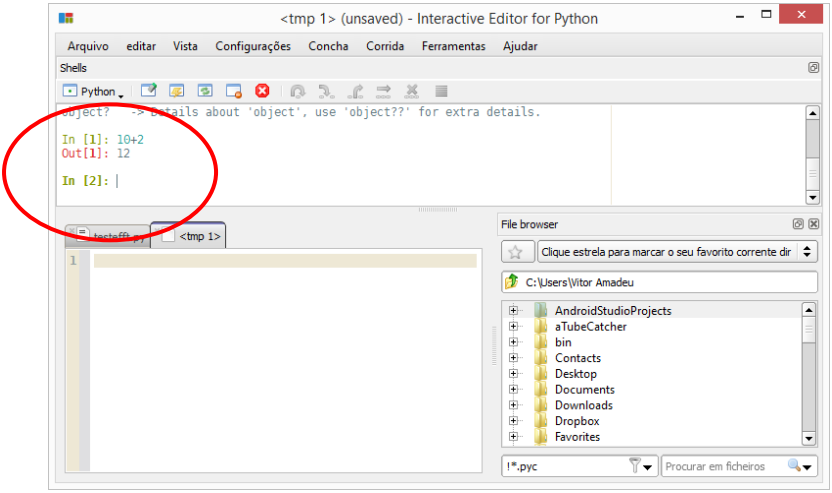
Outra possibilidade é usar a plataforma online chamada Google Colab, no qual é possível criar *notebooks* (cadernos) para executar e testar programas.

https://colab.research.google.com/

O Python é um software interpretado, ou seja, cada comando digitado no ambiente é logo executado após você pressionar o enter do teclado. Nos próximos tópicos, estaremos exercitando diversos exemplos no Python, de forma a entender como o mesmo funciona.

2. Operadores aritméticos

Podemos usar o Python como uma calculadora, bastando neste caso digitar diretamente a expressão matemática no mesmo, usando neste caso o prompt. Observe abaixo:



Note que ao digitar a expressão e pressionar o enter, o comando é imediatamente processado, tendo como resultado a soma da operação. Outra forma é escrever o programa como um script, salvá-lo e executá-lo em seguida, indo no menu Corrida-> Corrida arquivo como script.

O Python possui diversos operadores aritméticos, como os citados a seguir.

Operador	Função
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão de inteiro
**	Exponenciação

Observe abaixo outros exemplos no Python, baseado nos comandos apresentados anteriormente:

```
>>> 10 - 6
4
>>>
```

No exemplo acima temos uma operação de subtração sendo executada, onde a temos como resultado 4, que é a subtração de 10 – 6.

```
>>> 2 ** 3
8
>>>
```

Acima temos um exemplo de exponenciação, onde o valor 2 elevado a 3 dá como resultado 8.

```
>>> 100 % 3
1
>>>
```

Acima a operação é diferente, pois temos a apresentação do operador resto da divisão de inteiros (%) onde 100 dividido por 3 dá 33, porém com resto 1 como verificado acima.

3. Operadores lógicos

Faz uma das operações lógicas fornecendo como resultado verdadeiro (True) ou falso (False). Acompanhe um exemplo com os três operadores lógicos.

```
>>> x=1
>>> y=0
>>> x and y
0
>>> x or y
1
>>> not x
False
>>> not y
True
```

4. Operadores de bits (bitwise operators)

Faz a operação lógica bit a bit entre duas variáveis ou constantes. Acompanhe um exemplo.

```
>>> x=0x0F
>>> y=0xF0
>>> x & y
0
>>> x | y
255
>>> ~x
-16
>>> x ^y
255
>>> x>>1
7
>>> y>>4
15
```

5. Funções de conversão

Utilizadas para converter bases diferentes.

```
>>> x=10

>>> hex(x)

'0xa'

>>> bin(x)

'0b1010'

>>> oct(x)

'0o12'
```

6. Comentários

Os comentários são uma forma bem interessante de documentar o que o seu programa faz e assim facilitar a sua leitura posteriormente. Todos os comentários no Python são feitos colocando-se o operador # na frente. Observe o exemplo abaixo:

```
Type "copyright", "credits" or "license()" for more information.
>>> 10 % 3 #informa o resto da divisão de 10 por 3 em inteiros
1
>>>
```

Coloque comentários à vontade em seu programa, bastando inserir o caracter # antes de iniciá-lo. Observe que até a cor do texto altera-se para vermelho, para ficar claro que aquele texto é um comentário.

7. Variáveis

As variáveis são um importante meio para se guardar informações no Python. Não é necessário que estas sejam declaradas, bastando neste caso atribuir as variáveis o seu valor de operação. É importante frisar que o Python é case sensitive, ou seja, ele diferencia caracteres maiúsculos e minúsculos. Observe abaixo a definição de uma variável atribuindo a ela um valor do tipo string (conjunto de caracteres):

```
Type "copyright", "credits" or
>>> info="Treinamento Python"
>>>
```

Se quisermos ver o conteúdo desta variável basta que seja digitado o seu nome e em seguida, pressionado o botão Enter.

Observe outra definição de variável, neste caso uma variável inteira e decimal (float):

```
>>>
>>> var_inteira=10
>>> var_float=20.23
>>>
```

Para ver o conteúdo das variáveis o processo é o mesmo, bastando neste caso digitar o nome da variável e pressionar em

seguida o enter. É importante frisar que o Python é case sensitive e neste caso, a declaração da variável *var_float* seguida de outra variável chamada *Var_float* serão diferentes, ou seja, ocuparão diferentes regiões da memória.

Temos três tipos principais de variáveis, que são as variáveis string, float e integer. Na variável do tipo string armazenamos normalmente nomes, endereços, datas etc, ou seja, caracteres. Na variável do tipo float, armazenamos valores que contenham casas decimais, como por exemplo o valor de uma conta corrente. Na variável do tipo integer, valores inteiros como a idade de uma pessoa. Para sabermos o tipo de uma variável que está declarada no sistema, fazemos uso do comando type(nome_da_variável) como apresentado a seguir:

```
>>> var_inteira=10
>>> var_float=20.23
>>> type(var_float)
<class 'float'>
```

Após digitar type foi informado o nome da variável e em seguida apareceu o tipo que ela faz parte.

As variáveis declaradas não podem usar palavras reservadas da linguagem. O Python possui as seguintes:

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

8. Operadores Relacionais

Os operadores relacionais são usados para testar o conteúdo de variáveis para saber se, por acaso, alguma delas é igual, menor, maior etc. Veja abaixo a lista de operadores relacionais:

Operador	Função
==	Teste de Igualdade
!=	Diferente
>	Maior
<	Menor
>=	Maior ou Igual
<=	Menor ou Igual

Observe abaixo um exemplo para testarmos os operadores relacionais:

```
>>> a=2
>>> b=3
>>> a==b
False
```

Veja que no exemplo acima foram carregadas duas variáveis, sendo *a* com 2 e *b* com 3 e em seguida, usou-se o operador relacional == para checar se estas variáveis são iguais. O resultado foi False (Falso), como já era esperado. Observe a seguir outros testes condicionais feitos no Python:

```
>>> a=2
>>> b=3
>>> a==b
False
>>> a!=b     #a é diferente de b?
True
```

No exemplo acima é verificado se a é diferente de b, tendo como resultado true (verdadeiro).

```
>>> a=2
>>> b=3
>>> a==b
False
>>> a!=b     #a é diferente de b?
True
>>> a>b      #a é maior que b?
False
```

Já no exemplo anterior é usado o operador maior, obtendo o resultado falso.

```
>>> a<b      #a é menor que b?
True
```

Agora é usado o operador menor, tendo como resultado verdadeiro (true).

```
>>> a>=b
False
```

No exemplo acima foi verificado se a é maior ou igual a b, tendo como resultado falso.

```
>>> a<=b     #a é menor ou igual a b?
True
```

No exemplo acima foi testado se a é menor ou igual a b, tendo neste caso o resultado verdadeiro.

9. Trabalhando com strings

Faça o seguinte teste, atribua a variável *exemplo* o seguinte valor:

```
>>> exemplo="Elétrica"
```

Digite o nome da variável, neste caso *exemplo* e pressione enter, observe o que acontecerá.

```
>>> exemplo
'El\xe9trica'
```

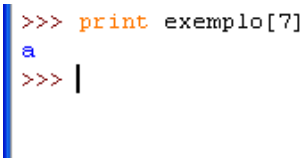
Note que o valor digitado a princípio é diferente do que foi atribuído inicialmente no programa. Isso ocorre por que o Python trabalha com caracteres ASCII e neste conjunto não há caracteres com acentuação. Para apresentarmos da forma correta a variável, devemos usar o comando print, como ilustrado a seguir:

```
>>> exemplo="Elétrica"
>>> exemplo
'El\xe9trica'
>>> print exemplo
Elétrica
```

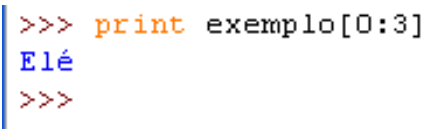
Podemos também apresentar individualmente os caracteres presentes na variável exemplo. Por exemplo, temos ao todo 8 caracteres nesta string (Elétrica possui 8 caracteres) e se quisermos mostrar a posição 0 desta variável, teremos o seguinte resultado:

```
>>> exemplo="Elétrica"
>>> exemplo
'El\xe9trica'
>>> print exemplo
Elétrica
>>> print exemplo[0]
E
```

A mesma ideia vale para as outras opções, como no exemplo abaixo:



É importante frisar que a contagem sempre inicia em 0, e no caso acima podemos ver dos caracteres de 0 a 7. É possível também imprimir uma faixa de caracteres, como apresentado no exemplo a seguir:



Observe no exemplo acima que foram impressos a partir do caracter 0 o total de 3 caracteres, que neste caso teve o resultado visto acima.

Para concatenar (unir) duas ou mais strings utiliza-se o operador +.

```
>>> s1="Rio "

>>> s2=" de "

>>> s3=" Janeiro"

>>> s1+s2+s3
```

```
'Rio  de  Janeiro'
```

Como a string é um objeto, para deixar todos os caracteres maiúsculos utiliza-se o método upper().

```
>>> t1="brasil"
>>> t1.upper()
'BRASIL'
```

A capitulação, ou seja, deixar o primeiro caractere maiúsculo é feito através do método capitalize().

```
>>> t1="brasil"
>>> t1.capitalize()
'Brasil'
```

Acompanhe o próximo exemplo que captura um caracter de uma string.

```
>>> txt="RJ"
>>> txt[0]
'R'
>>> 2*txt[0]
```

```
'RR'
```

10. O comando If

O IF (se) é usado para testarmos condições e variáveis dentro do Python. Para usarmos o IF usaremos com frequência os operadores relacionais, que foram apresentados em tópicos passados. Observe o exemplo abaixo, em que é declarado um valor a uma variável e depois é verificado se a variável possui um valor maior que o apresentado no exemplo.

```
>>> a=2
>>> if a>10:
        print "A é maior que 10"
else:
        print "A é menor ou igual a 10"


A é menor ou igual a 10
```

Note no exemplo acima que temos inicialmente o IF testando se a variável a é maior que 2. Caso seja, o comando em seguida será executado, mostrando a mensagem *"A é maior que 10"*. Caso este teste não seja verdadeiro, o segundo comando será executado, neste caso o else (senão) onde irá mostrar a mensagem *"A é menor ou*

igual a 10”. Verifique que o esperado ocorreu, executando neste caso o else e mostrando a mensagem que está no exemplo acima.

Temos também o comando elif (else IF) que é uma forma de testar mais vezes o conteúdo de uma variável. Acompanhe no exemplo abaixo a aplicação deste comando:

```
>>> a=10
>>> if a>10:
        print "A é maior que 10"
elif a==10:
        print "A é igual a 10"
else:
        print "A é menor que 10"


A é igual a 10
>>>
```

Podemos ter vários elif (else IF) em nosso programa, tornando assim um comando muito útil para quando precisarmos realizar diversos testes em uma variável do sistema.

11. O comando while

Sempre que precisarmos repetir um bloco de comandos repetidas vezes, podemos usar o comando *while* (enquanto).

Acompanhe no exemplo abaixo um demonstrativo do uso deste comando.

```
>>> x=1
>>> while x<=20:
        print x
        x=x+1


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
>>>
```

Verifique que inicialmente fazemos a variável *x* igual a 1 e em seguida, no comando *while*, repetimos os dois comandos que estão abaixo que é o de impressão do conteúdo da variável e do seu incremento, até que a condição escrita no while passe a ser falsa, ou seja, o conteúdo da variável x seja maior que 20.

12. O comando for

O comando *for* no geral tem a mesma função do while, que é repetir o código até que uma determinada condição passe a ser falsa. Leia no exemplo abaixo o uso do for e note que ele faz a mesma função do comando *while* apresentado acima.

```
>>> for x in range(1,20):
        print x

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

Podemos variar a faixa do *range* e assim executar um número diferente de loops no programa. Para alterar o passo basta informar um terceiro parâmetro:

```
>>> for x in range(1,20,2):
        print(x)
1
3
5
7
9
11
13
15
17
19
```

Outra forma é imprimir de acordo com a definição de um vetor.

```
>>> for x in [1,2,3,4,5]:
        print(x)
1
2
3
4
```

5

Uma forma de imprimir uma faixa de uma equação:

```
>>> [x**2 for x in [1,2,3]]
[1, 4, 9]
```

Ou um par ordenado.

```
>>> [(x,x**2) for x in range(5)]

 [(0, 0), (1, 1), (2, 4), (3, 9), (4, 16)]
```

13. Usando listas

As listas são uma forma de se guardar dados para acessá-los através de um índice. No exemplo abaixo, é criado uma lista de três elementos e em seguida é visualizado o conteúdo de cada um destes elementos:

```
>>> lista=["Python","Programação","Linguagem"]
>>> print lista[0]
Python
>>> print lista[1]
Programação
>>> print lista[2]
Linguagem
>>>
```

Usando as listas não foi necessário criar uma variável para armazenar cada um dos valores e para acessar mais informações bastou informar o índice e ter acesso assim ao dado gravado naquela região de memória. Existem algumas funções interessantes intrínsecas as listas que veremos em seguida, que faz o uso deste recurso bastante interessante.

O comando insert permite inserir novas informações a lista. Observe o exemplo abaixo para compreender o funcionamento deste recurso.

```
>>> lista=["Python","Programação","Linguagem"]
>>> lista.insert(3,"Debug")
>>> print lista
['Python', 'Programa\xe7\xe3o', 'Linguagem', 'Debug']
>>>
```

Note no exemplo acima que inserimos na terceira posição um novo conteúdo e em seguida pudemos imprimir e ver o resultado esperado. Com o comando insert podemos inserir novos dados a lista em qualquer posição, conforme o exemplo apresentado abaixo:

```
>>> lista=["Python","Programação","Linguagem"]
>>> lista.insert(1,"Debug")
>>> print lista
['Python', 'Debug', 'Programa\xe7\xe3o', 'Linguagem']
>>>
```

O comando *append* permite adicionar informações a lista, porém em seu final, como no exemplo a seguir.

```
>>> lista=["Python","Programação","Linguagem"]
>>> lista.append("Teste")
>>> print lista
['Python', 'Programa\xe7\xe3o', 'Linguagem', 'Teste']
>>>
```

Há outras funções que estão listadas abaixo. É importante que o leitor procure testar cada uma delas de forma a entender perfeitamente o seu funcionamento.

Função	Descrição
count()	Conta a quantidade de vezes que um elemento aparece na lista
index()	Retorna a posição de determinado elemento
remove()	Remove o primeiro elemento da lista
reverse()	Inverte a ordem dos elementos em uma lista
sort()	Ordena os elementos de uma lista

14. Conhecendo as Tuplas

As tuplas tem a mesma funcionalidade das listas, sendo a diferença principal o fato destas não poderem ser alteradas após a sua criação. Verifique o exemplo apresentado a seguir.

```
>>> tuplas=(10,20,30)
>>> tuplas
(10, 20, 30)
>>> tuplas[0]=100

Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    tuplas[0]=100
TypeError: 'tuple' object does not support item assignment
>>>
```

Note pelo exemplo acima que ao tentar alterar o conteúdo da tupla ocorreu um erro.

15. Dicionários

Diferente de uma tupla e lista que possuem índice no dicionário a indexação é feita por strings formando um contêiner, como sugere o exemplo.

```
>>> aurelio={'RJ':'Estado da região sudeste','Apple':'Empresa de tecnologia estadunidense','Roma':'Capital Italiana','Trem':'Meio de transporte inventado no século XIX'}
```

```
>>> aurelio

     {'RJ': 'Estado da região sudeste', 'Apple': 'Empresa de
tecnologia estadunidense', 'Roma': 'Capital Italiana', 'Trem':
'Meio de transporte inventado no século XIX'}

     >>> aurelio['RJ']


     'Estado da região sudeste'


     >>> aurelio['Apple']


     'Empresa de tecnologia estadunidense'
```

16. Conjuntos

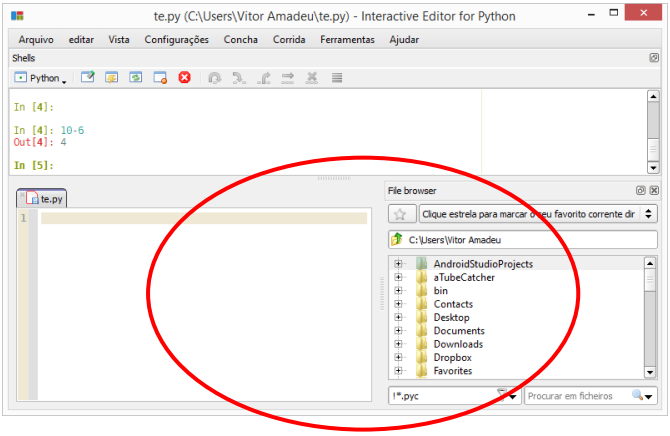
Acompanhe um exemplo que demonstra as operações diferença, união, interseção e diferença simétrica.

```
>>> a={1,2,3}
>>> b={2,3,4}
>>> a-b
{1}
>>> a | b
{1, 2, 3, 4}
```

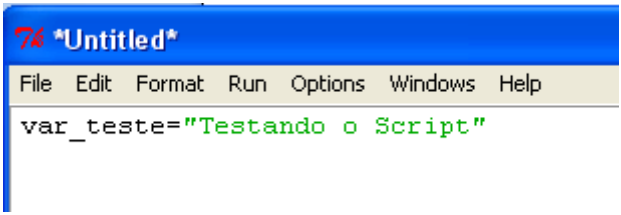
```
>>> a & b
{2, 3}
>>> a ^ b
{1, 4}
```

17. Criando scripts

O Python possui um recurso que permite criar nossos scripts para podermos usá-lo em seguida. Por exemplo, baseado nos exemplos feitos anteriormente se fecharmos o Python e abri-lo novamente, todas as definições serão perdidas. Agora se escrevermos tudo em um script (arquivo texto) teremos todas as informações salvas e bastará neste caso importar o script para que venha a funcionar no Python. Para criar um script, com o Python aberto observe o campo marcado a seguir.

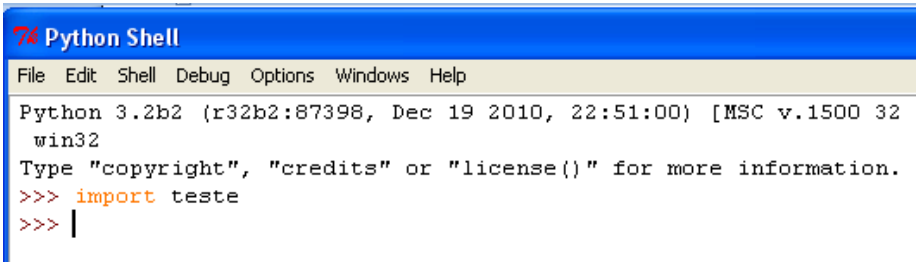


Vamos fazer como exemplo inicial a definição de uma variável, como mostrado a seguir.

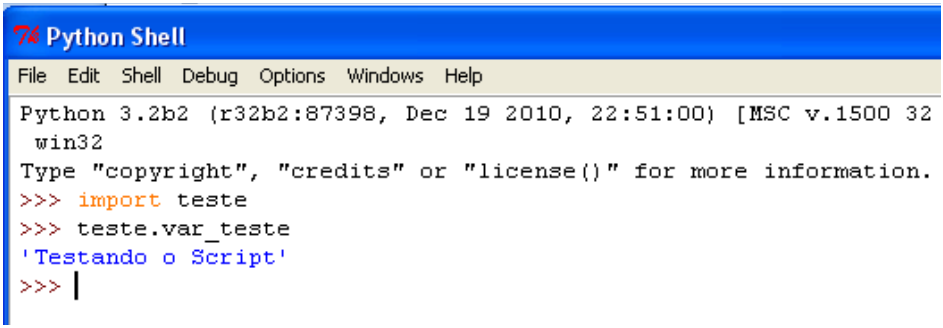


Agora salve o script, indo no menu Arquivo -> Salve. Salve na pasta padrão apresentada com o nome teste.py.

Volte a janela principal do Python (Shell) e faça a importação deste script para o Python, digitando:



Faça o que está apresentado abaixo e veja o resultado.



Note que digitamos o nome do script (teste) seguido do nome da variável declarada dentro deste script, que foi a variável var_teste. Em seguida, foi apresentado o valor que carregamos dentro daquele script. Faça o teste e declare outras variáveis de forma a comprovar o funcionamento dos scripts. Crie uma nova script e escreva as seguintes funções:

```
def perimetro(r):
    try:
        r=float(r)
        return 2*3.14*r
    except:
        print ('O argumento da funcao deve ser um numero.')

def area(r):
    try:
        r=float(r)
        return 3.14*(r**2)
    except:
        print ('O argumento da funcao deve ser um numero.')

def potencia(x,y):
    try:
        return x**y
    except:
        print ('Argumentos invalidos')
```

Salve o arquivo com o nome mat.py. Importe seu conteúdo para o sheell digitando importe mat. Chame a função através de mat.area(3) e veja o resultado. Outra forma de obter o mesmo fim porém sem precisar digitar o nome do módulo está apresentado abaixo.

```
>>> from mat import area
>>> area(3)
28.26
```

Para importar todas as funções digite:

```
from mat import *
```

18. PI e número de Euler

O exemplo abaixo obtém o valor de PI e número de Euleratravés do módulo math.

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

19. Funções matemáticas

Acompanhe os exemplos obtidos após importar o módulo math.

```
>>> sin(pi)
1.2246467991473532e-16
>>> cos(pi)
-1.0
>>> log(1)
0.0
>>> log10(10)
1.0
>>> pow(3,2)
9.0
```

20. Obtendo a data e hora

Importando a biblioteca time podemos obter a hora local do PC.

```
>>> import time

>>> time.asctime()

'Tue Mar 19 19:38:05 2019'

>>> time.localtime()
time.struct_time(tm_year=2019, tm_mon=3, tm_mday=19,
tm_hour=19, tm_min=38, tm_sec=10, tm_wday=1, tm_yday=78,
tm_isdst=0)
```

21. Calculando o tempo para executar uma rotina

Importando a biblioteca time é possível verificar a diferença de tempo para tratar uma rotina.

```
from time import *

from math import *

t0=time()

print(sin(pi))

t1=time()-t0

print(str(t1) + " s")
```

22. Emitindo som

Importando a biblioteca winsound podemos emitir som, no qual o primeiro parâmetro é a frequência e o segundo o tempo.

```
>>> import winsound
>>> beep=winsound.Beep
>>> beep(1000,1000)
```

23. Calendar

Importando a biblioteca calendar realizar operações com o calendário. O próximo exemplo verifica se um ano é bissexto.

```
>>> import calendar
>>> calendar.isleap(2020)
True
```

O exemplo abaixo imprime os dias de um mês.

```
>>> calendar.prmonth(2019,3)
     March 2019
Mo Tu We Th Fr Sa Su
             1  2  3
```

```
 4  5  6  7  8  9 10

11 12 13 14 15 16 17

18 19 20 21 22 23 24

25 26 27 28 29 30 31
```

24. Números complexos

A seguir exemplos com números complexos.

```
>>> z1=1+2j

>>> z2=3+4j

>>> z1+z2

(4+6j)

>>> z1-z2

(-2-2j)

>>> z1*z2

(-5+10j)

>>> z1/z2

(0.44+0.08j)
```

25. Funções Pré-Definidas

O Python disponbiliza uma série de funções que vem agregadas na própria ferramenta. Veremos algumas delas agora, porém não

deixe de ver em seguida o help do ambiente pressionando F1, pois há outras funções muito importantes dispostas na ferramenta.

Vejamos inicialmente o funcionamento da função *range*, que retorna os valores definidos entre o valor inicial e final sem incluir o valor final. A seguir um exemplo desta função:

```
>>> range(1,10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A função *len*, retorna a quantidade de caracteres de uma string, conforme ilustra o próximo exemplo.

```
>>> palavra="Python"
>>> len(palavra)
6
```

Vejamos a função *list*, que converte uma string em uma lista. Observe o exemplo apresentado:

```
>>> palavra="Python"
>>> list(palavra)
['P', 'y', 't', 'h', 'o', 'n']
```

A função *tuple* permite converte a string em uma tupla, observe o exemplo:

```
>>> tuple(palavra)
('P', 'y', 't', 'h', 'o', 'n')
```

A função *int* permite converte um dado do tipo float em um inteiro, conforme o próximo exemplo.

```
>>> var_float=50.23
>>> int(var_float)
50
```

Já a função *float* faz o inverso, convertendo um dado do tipo inteiro para float. A seguir o exemplo.

```
>>> var_int=50
>>> float(var_int)
50.0
```

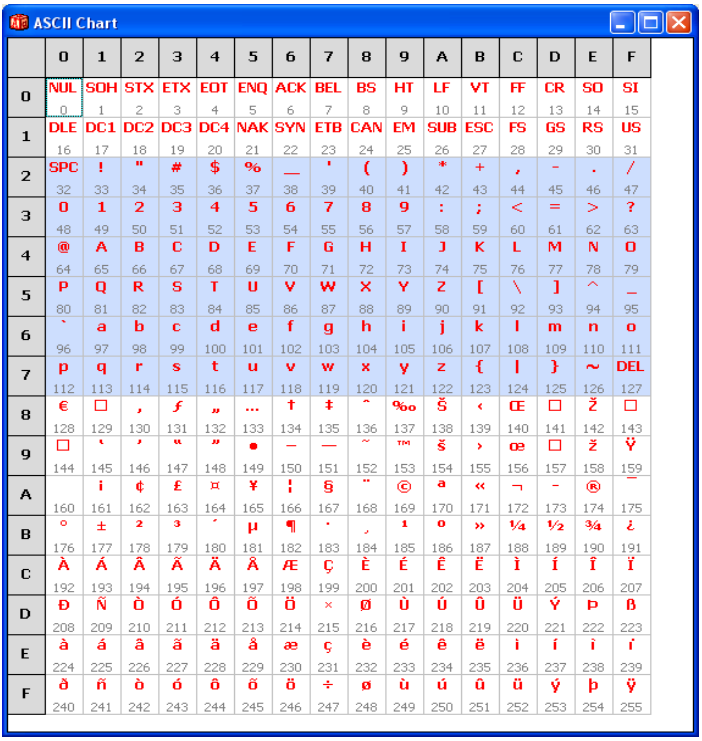
A função *str* permite converte qualquer tipo de dado para um valor do tipo string. Acompanhe o exemplo abaixo:

```
>>> var_int=50
>>> str(var_int)
'50'
```

A função *chr* apresentada a seguir permite converter um valor numérico decimal para o seu equivalente em ASCII. Abaixo temos um exemplo:

```
>>> chr(65)
'A'
```

A tabela ASCII está ilustrada na próxima figura.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
1	DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
2	SPC 32	! 33	" 34	# 35	$ 36	% 37	_ 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127
8	€ 128	□ 129	‚ 130	ƒ 131	„ 132	… 133	† 134	‡ 135	ˆ 136	‰ 137	Š 138	‹ 139	Œ 140	□ 141	Ž 142	□ 143
9	□ 144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	˜ 152	™ 153	š 154	› 155	œ 156	□ 157	ž 158	Ÿ 159
A	160	¡ 161	¢ 162	£ 163	¤ 164	¥ 165	¦ 166	§ 167	¨ 168	© 169	ª 170	« 171	¬ 172	- 173	® 174	¯ 175
B	° 176	± 177	² 178	³ 179	´ 180	µ 181	¶ 182	· 183	¸ 184	¹ 185	º 186	» 187	¼ 188	½ 189	¾ 190	¿ 191
C	À 192	Á 193	Â 194	Ã 195	Ä 196	Å 197	Æ 198	Ç 199	È 200	É 201	Ê 202	Ë 203	Ì 204	Í 205	Î 206	Ï 207
D	Ð 208	Ñ 209	Ò 210	Ó 211	Ô 212	Õ 213	Ö 214	× 215	Ø 216	Ù 217	Ú 218	Û 219	Ü 220	Ý 221	Þ 222	ß 223
E	à 224	á 225	â 226	ã 227	ä 228	å 229	æ 230	ç 231	è 232	é 233	ê 234	ë 235	ì 236	í 237	î 238	ï 239
F	ð 240	ñ 241	ò 242	ó 243	ô 244	õ 245	ö 246	÷ 247	ø 248	ù 249	ú 250	û 251	ü 252	ý 253	þ 254	ÿ 255

A função *ord* faz o inverso, convertendo um caractere para o seu valor numérico na tabela ASCII. Acompanhe o exemplo apresentado abaixo.

```
>>> ord('A')
65
>>>
```

A função *hex* permite converter um valor decimal para hexadecimal. Abaixo temos um exemplo do uso desta função:

```
>>> hex(10)
'0xa'
```

Já a função *bin* converte para binário, como apresentado a seguir:

```
>>> bin(10)
'0b1010'
```

A função *oct* permite converter para octal, observe o exemplo abaixo:

```
>>> oct(10)
'012'
```

A função *unichr* permite converter um valor numérico de 0 a 65535 em seu representante na tabela UNICODE. Observe abaixo um exemplo:

```
>>> unichr(100)
u'd'
```

A função *pow* permite com que seja calculado a potência de um número, como mostrado a seguir:

```
>>> pow(10,2)
100
```

A função *min* e *max* retornam o mínimo e máximo respectivamente, mediante dois valores passados para a função. Acompanhe abaixo o exemplo:

```
>>> min(100,2)
2
>>> max(100,2)
100
```

A função *abs* permite que seja calculado o valor absoluto de um número, ou seja, seu módulo, apenas a parte positiva. No experimento abaixo podemos ver o uso de tal função:

```
>>> valor=-100
>>> abs(valor)
100
```

26. Comando type

Permite saber o tipo de dado de uma variável

```
>>> type("Olá")

<class 'str'>

>>> type(3.1)

<class 'float'>

>>> type (3)

<class 'int'>

>>> type(1+2j)

<class 'complex'>
```

27. Entrada de dados

A entrada de dados é feita através do comando input(). Crie uma nova script e escreva o código abaixo.

```
print("Informe sua idade")
```