**Preparatório para Exame 98-381**

**Introdução a programação usando Python**

[**herickfreitas@hotmail.com**](mailto:herickfreitas@hotmail.com)

**27/03/2018**

Sumário – Roteiro de aprendizado

[**1. Introdução. 3**](#_Toc511146913)

[**2. Exibição de texto. 8**](#_Toc511146914)

[**3. Variáveis de cadeia de caracteres. 10**](#_Toc511146915)

[**4. Armazenamento de números. 14**](#_Toc511146916)

[**5. Como trabalhar com datas e horas. 19**](#_Toc511146917)

[**6. Tomando decisões com o código. 22**](#_Toc511146918)

[**7. Decisões complexas. 25**](#_Toc511146919)

[**8. Repetição de eventos. 29**](#_Toc511146920)

[**9. Repetição de eventos até a conclusão. 34**](#_Toc511146921)

[**10. Lista de lembranças. 37**](#_Toc511146922)

[**11. Como salvar informações em arquivos. 39**](#_Toc511146923)

[**12. Leitura a parti de arquivos 45**](#_Toc511146924)

[**13. Funções. 47**](#_Toc511146925)

[**14. Manipulação de erros. 50**](#_Toc511146926)

[**15. Extras 55**](#_Toc511146927)

# Introdução.

**Qual o público alvo deste curso?** Qualquer pessoa interessada em aprender, estudante, profissionais de TI, profissionais de Ciências Exatas.

**Por que escolher Python?** Python possui uma sintaxe de baixa complexidade, é gratuito, aplicação leve, possui uma infinidade de recursos e aplicações...

**Qual software iremos usar para programar?** Visual Studio Community, é gratuito e pode ser encontrado facilmente ne web para download.

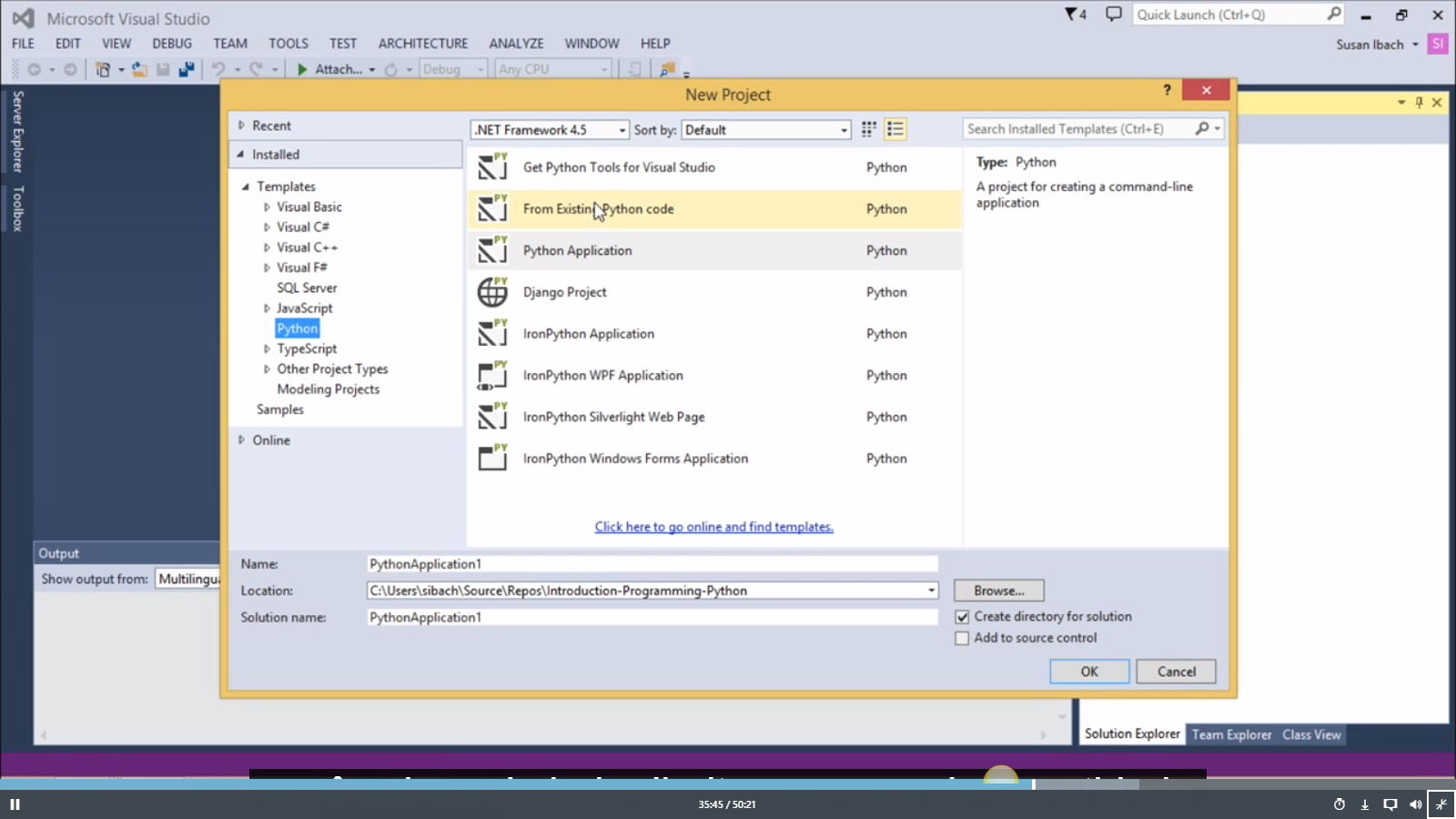
**Qual finalidade de um programa?** Um programa resolve problemas, facilitando a vida.

**Onde Python é usado?** O Python é muito utilizado para coleta de dados, mineração de dados, robótica, edição de filmes, construção de sites, etc...

**Como é o processo de instalação?** Pesquise me algum site de busca “download python for visual studio free”, será direcionado para um portal da Microsoft onde será disponibilizado algumas versões para download. Baixe a versão Community e selecione a adição do módulo de programação em Python. Estou usando a versão 2017.



Após a instalação crie um novo projeto em Python. Selecione Python application. Crie um projeto chamado HelloWorld

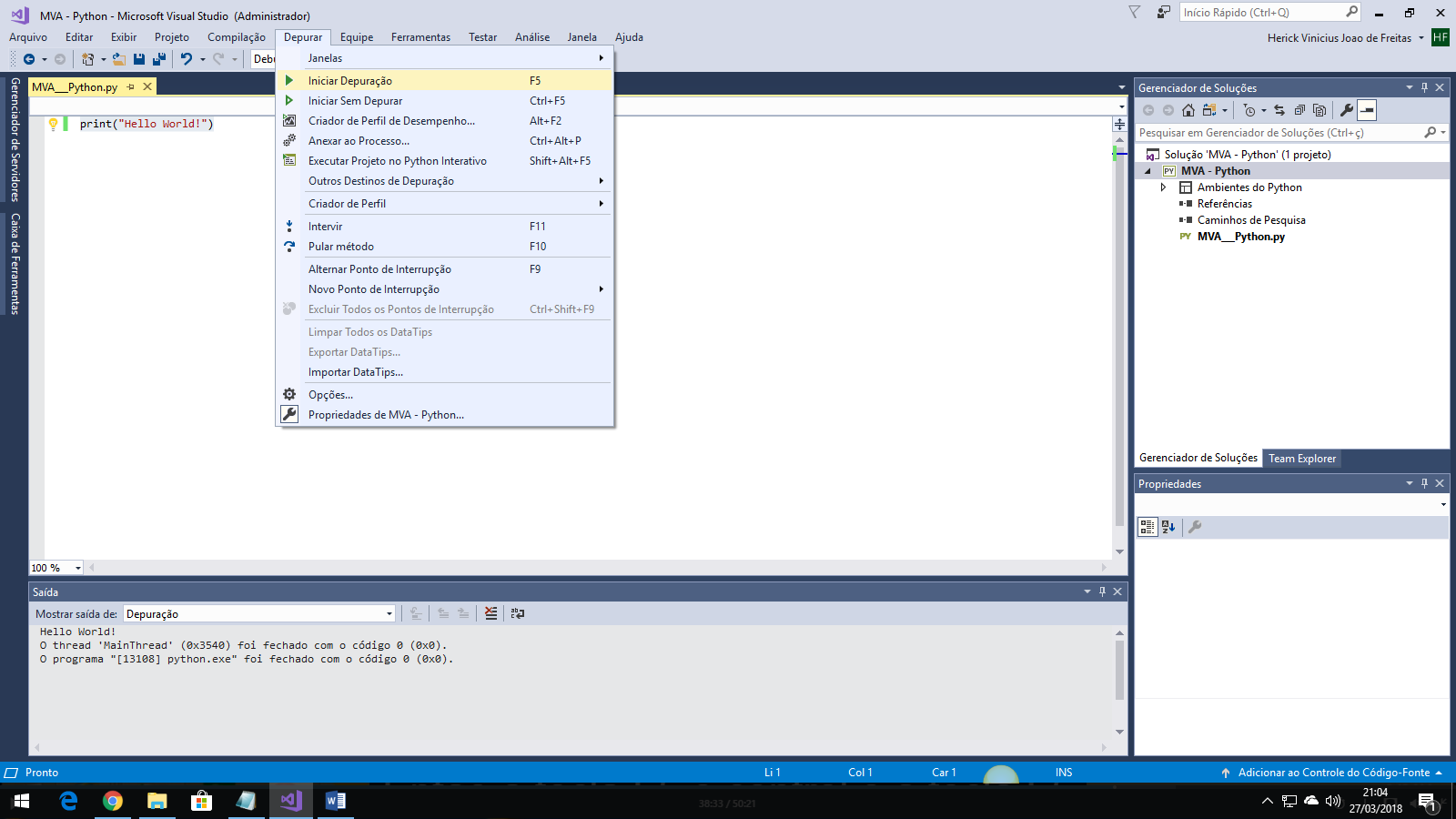


Será criado um arquivo com extensão .py que é a extensão padrão Python.

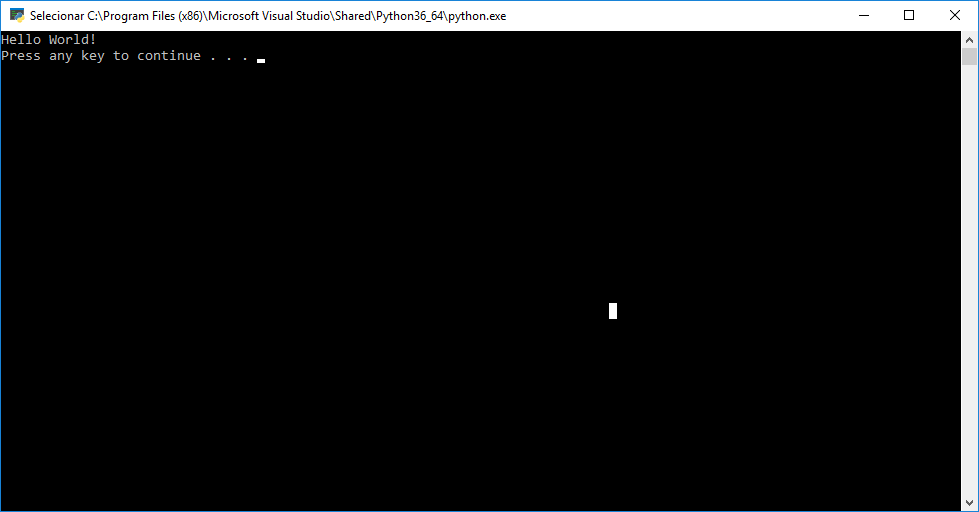
Crie um código para impressão da mensagem Hello World!.

print("Hello World!")

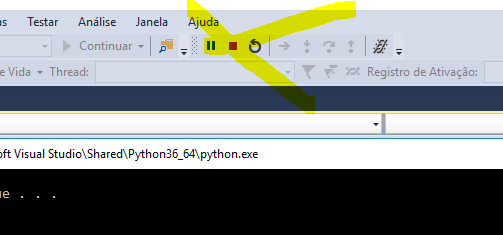
O código poderá ser executado com F5 para que ocorra a depuração da programação.



Como o código representa somente uma saída de dados, será apresentada a mensagem.



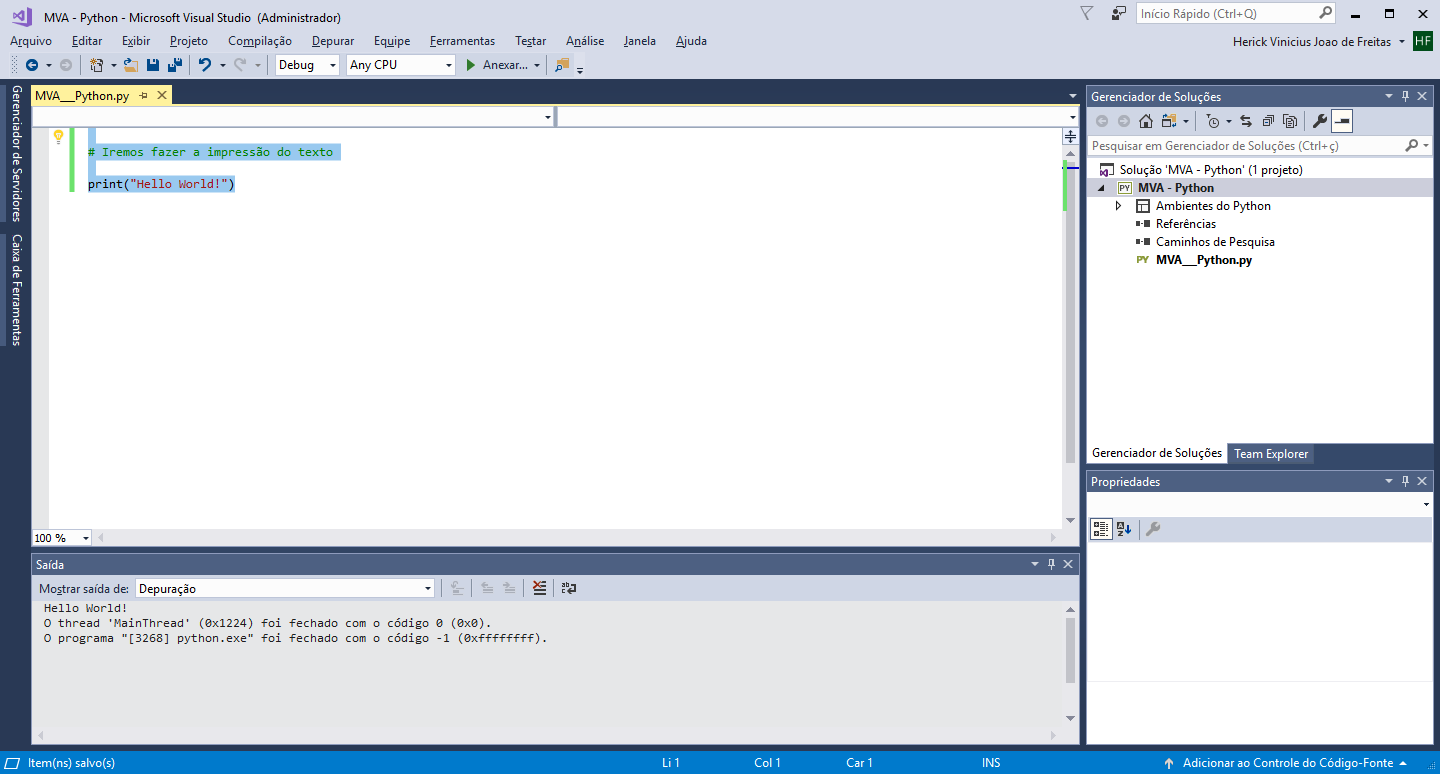
Para encerrar a execução do programa você pode fechar a tela dar Enter ou clicar o Stop na barra superior.



Desenvolva bons hábitos desde o início, comente sempre seu código.

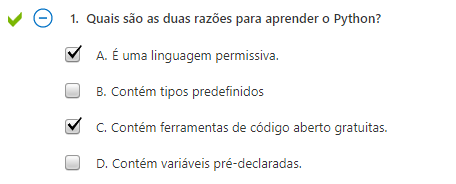
# Iremos fazer a impressão do texto

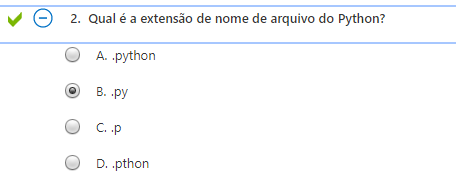
print("Hello World!")

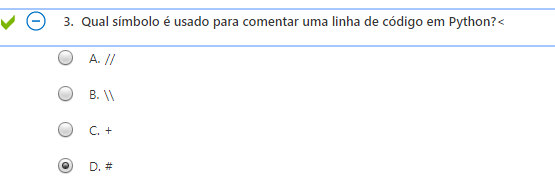


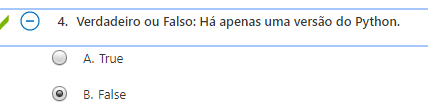
Use o início rápido no canto superior direito, para fazer pesquisas e sanar dúvidas sobre sua programação.

**Exercícios:**











# Exibição de texto.

O Python possui uma abordagem muito simples quando trabalhamos com textos, basicamente um texto é identificado quando está entre aspas(‘ ‘) ou aspas duplas(“ “). Então conforme o projeto Olá mundo quando associamos um texto ao comando print() efetuamos uma impressão.

Efetuando impressão de várias linhas, o comando print() fará uma impressão contínua de todo o texto contido entre parênteses. Então uma maneira menos prática seria uma print() por cada linha que deseja imprimir.

print("Além destas, outras cinco pessoas são alvos de prisão temporária – que valem por cinco dias, prorrogáveis por mais cinco. No total,")

print("dos 11 pedidos de prisão, seis foram expedidos a professores. A operação cumpre, ainda, 12 mandados de busca a apreensão.")

Uma segunda opção e mais prática é usar um texto inteiro em um só print(), efetuando a quebra de linha com \n.

print("Além destas, outras cinco pessoas são alvos de prisão temporária – que valem por cinco dias, prorrogáveis por mais cinco. No total,\n dos 11 pedidos de prisão, seis foram expedidos a professores. A operação cumpre, ainda, 12 mandados de busca a apreensão.")

Também há a opção de informar o texto entre aspas triplas, tanto faz se é simples ou duplas. Assim o texto será impresso exatamente com a quebra de linha que for escrito.

print("""Além destas, outras cinco pessoas são alvos de prisão temporária – que valem por cinco dias, prorrogáveis por mais cinco. No total,

dos 11 pedidos de prisão, seis foram expedidos a professores. A operação cumpre, ainda, 12 mandados de busca a apreensão.""")

Um outro recurso é concatenar texto com ( + ), assim dois ou mais textos podem ser representados de maneira única.

print("Além destas, outras cinco pessoas" + "são alvos de prisão temporária – que valem por cinco dias, prorrogáveis por mais cinco.")

Há um recurso que pode ser aplicado quando precisamos usar aspas duplas e simples no mesmo texto. Use \ antes das aspas para que elas sejam abordadas como texto.

print("Além destas, outras cinco pessoas \"são alvos de prisão temporária\" – que valem por cinco dias, prorrogáveis por mais cinco.")

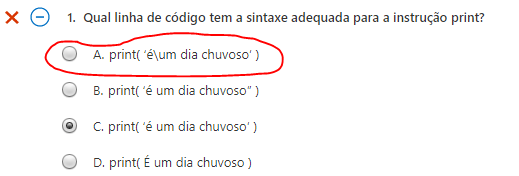
Se precisar aplicar uma tabulação no texto use \t

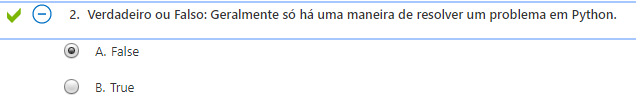
print("2kg\tFeijão")

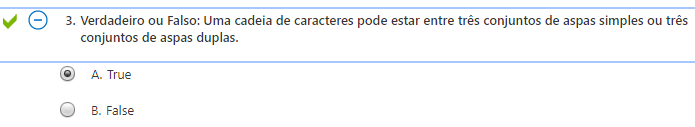
Quando precisar inserir no texto alguma palavra com \ antecedendo use \\ para não gerar quebra ou tabulação no texto.

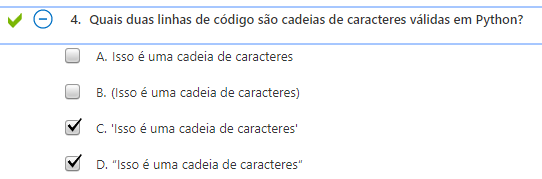
print("Acesse o site usando [\\news](file:///\\news)")

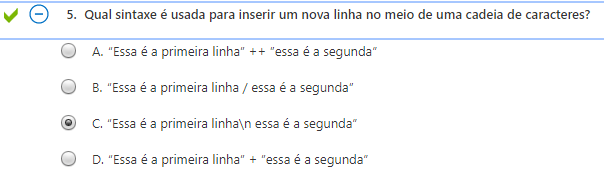
**Exercícios:**











# Variáveis de cadeia de caracteres.

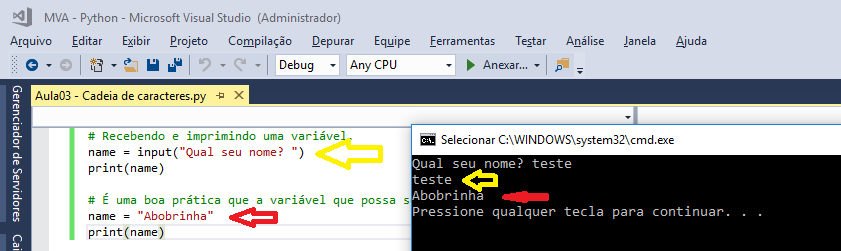
Toda programação parte de uma ação ou interação com mundo exterior, seja através de uma entrada de dado textual ou coleta de dados através de imagens, áudios, websites, IoT, robótica, etc....

Então, esta entrada se dá através de um input de informações, este input será armazenado em uma variável e esta variável será utilizada pelo programa.

# Recebendo e imprimindo uma variável.

name = input("Qual seu nome? ")

print(name)



É muito importante que se uma variável possa ser usada mais de uma vez ela deva ser limpa ou resetada para que não ocorra erros.

As regras de criação de variáveis são: Não pode conter espaços, CaseSensitive e não pode começar com número.

Boas práticas:

Tenha objetividade na escolha de variáveis exemplo: Rua já referencia o cadastro da variável rua, não use RuaDoCliente ou RDC.

Use variáveis com nomes que as referenciem evite X, Y, etc... Uma variável pode conter até 82 caracteres. Escolha um esquema como padrão, use letras maiúsculas ou minúsculas, exemplo DataNascimento ou datanascimento.

Trabalhando com variáveis.

Vamos criar duas variáveis e concatena-las.

nome = input("Qual seu nome? ")

pais = input("Qual Pais você vive? ")

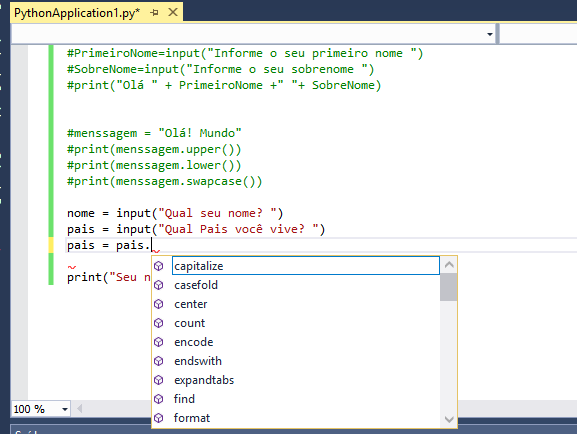
pais = pais.upper()

print("Seu nome é " + nome + " e você vive em " + pais)

Funções de cadeias de caracteres

Serve para manipular o texto, alterando para maiúscula ou minúscula, dentre outros recursos.

O Visual Studio possui recurso de sugestão durante a digitação do código chamado de IntelliSense, assim você receberá algumas dicas dos recursos disponíveis ou use Ctrl + J ou Ctrl + SPACE



Não é possível memorizar todos os recursos, então use as ferramentas disponíveis. F1 é o help, use o intellisense, documentação, sites de busca, etc...

Uma boa prática de programação é sempre declarar as variáveis. Informando-a(as) zerada(s) ou vazia(s).

nome = ""

pais = ""

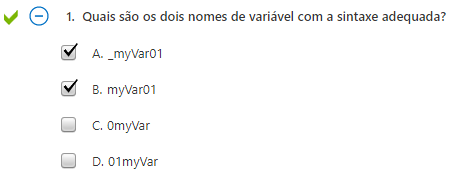
nome = input("Qual seu nome? ")

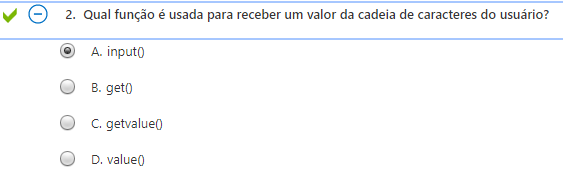
pais = input("Qual Pais você vive? ")

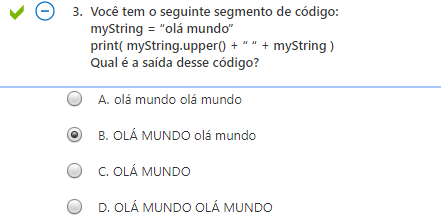
pais = pais.upper()

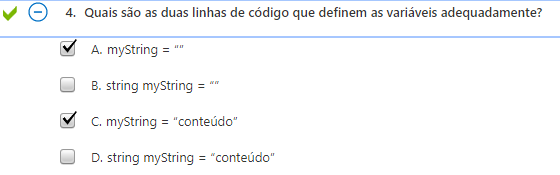
print("Seu nome é " + nome + " e você vive em " + pais)

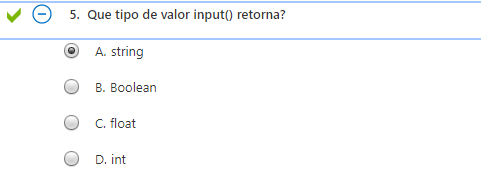
**Exercícios:**





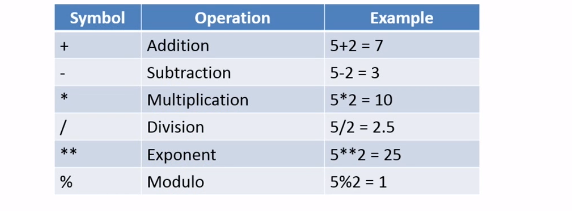




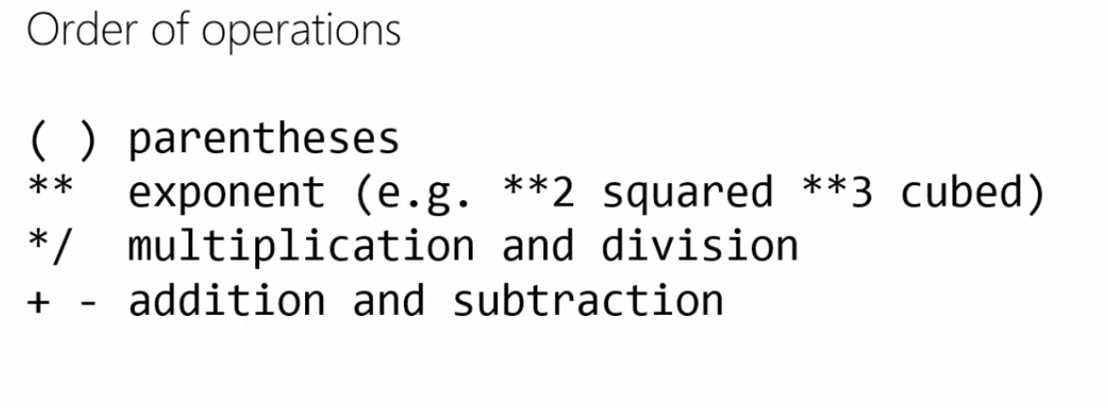


# Armazenamento de números.

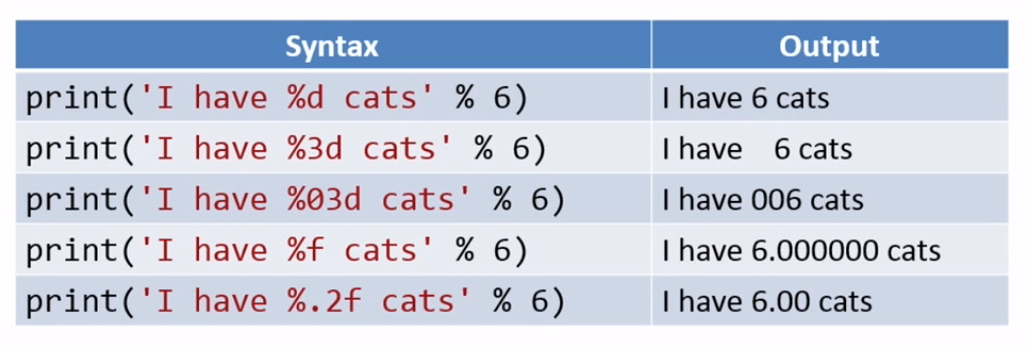
Assim como possuímos variáveis do tipo String, também possuímos variáveis numéricas. Certamente necessária para maior parte das aplicações de programação em Python. De manera nativa o Python reconhece os operadores matemáticos e realiza suas operações.



Durante a execução das equações matemáticas há uma ordem de execução, segue uma representação de como o Python trata essas operações.



Então não será eventual a necessidade de associarmos na saída de dados informações referentes a variáveis de tipos diferentes, além disso representando informações numéricas como inteiros ou pontos flutuantes, etc...



Salvo que na representação a informação d representa decimal e f representa ponto flutuante.

# Manipulação de números

# Calculando área de um triangulo

area = 0

largura = 10

altura = 10

area = (largura \* altura) / 2

# impressão decimal

print("A área do tringulo é %d" %area)

# impressão flutuante

print("A área do tringulo é %f" %area)

# impressão flutuante com duas casas decimais

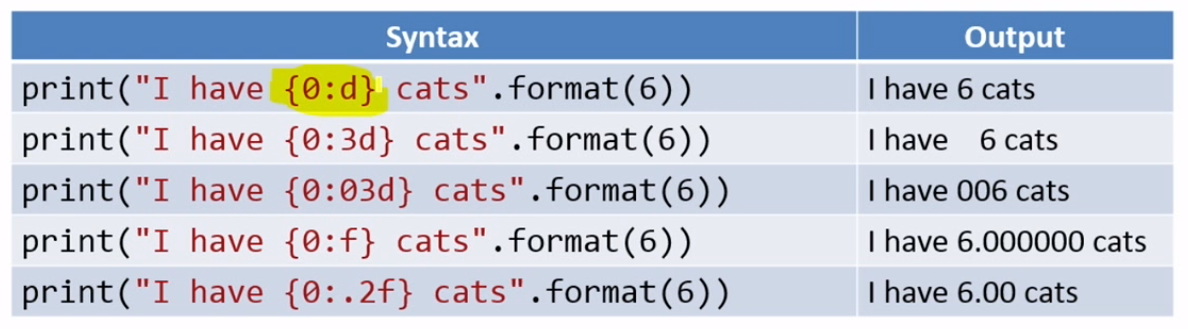
print("A área do tringulo é %.2f" %area)

# impressão preenchendo com zeros a esquerda, muito útil para

# relatórios, refere-se a quantidade de espaços.

print("A área do tringulo é %03d" %area)

Há outro método de fazer a impressão, este método foi disponibilizado a partir da versão 3 do Python.



O {0: presente em todas impressões referência que é o primeiro item dentro do .format().

# Manipulação de números

# Calculando área de um triangulo usando format

area = 0

largura = 10

altura = 10

area = (largura \* altura) / 2

# impressão decimal

print("A área do tringulo é {0:d}".format(500))

# impressão flutuante

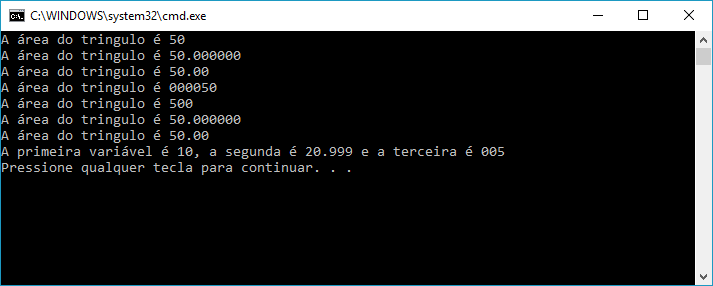
print("A área do tringulo é {0:f}".format(area))

# impressão flutuante com duas casas decimais

print("A área do tringulo é {0:.2f}".format(area))

# Agora iremos validar a impressão com multimas variáveis.

print("A primeira variável é {0:d}, a segunda é {1:.3f} e a terceira é {2:03d}".format(10,20.999,5))



Atenção: Talvez esteja escrevendo um comando muito extenso que está ficando muito grande em uma linha, use \ barra invertida e continue na linha a seguir, Python irá considerar que se trata de um comando único.

Total = 2 + 4 \* 40 / 30 + 50 \

40 + 10 – 20

Também é possível quebrar a sintaxe em duas. É muito importante que a linha seguinte possui tabulação com a superior, assim o código fará uma associação.

# Agora iremos validar a impressão com multiplas variáveis.

print("A primeira variável é {0:d}, a segunda é {1:.3f} e " +

"a terceira é {2:03d}".format(10,20.999,5))

Operações matemáticas

Quando formos realizar operações matemáticas é muito importante classificar as variáveis, pois senão serão tratadas como String.

# Agora iremos representar operações matemáticas.

# É muito importante classificar suas variáveis durante o processo de calculo.

salario = input("Informe seu salário " )

comissao = input("Informe sua comissão ")

# Valor não classificado

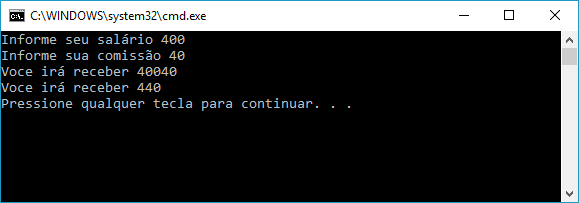
cheque = salario + comissao

print("Voce irá receber {0}".format(cheque))

# Valor classificado

chequeFinal = int(salario) + int(comissao)

print("Voce irá receber {0}".format(chequeFinal))



A questão anterior foi solucionada, mas não da maneira ideal. A condição ideal era que o dado fosse tratado na entrada, assim ficaria armazenado no formato correto.

# Reescrevendo o código usando boas práticas.

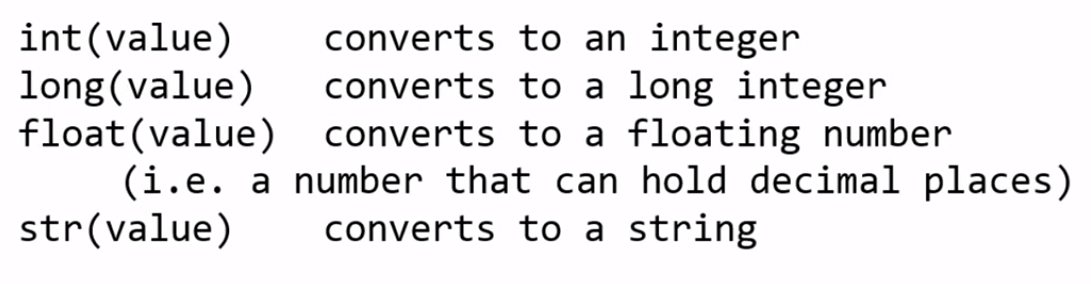
salarioInt = float(input("Informe seu salário " ))

comissaoInt = float(input("Informe sua comissão "))

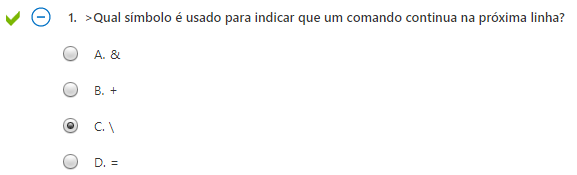
chequeInt = salarioInt + comissaoInt

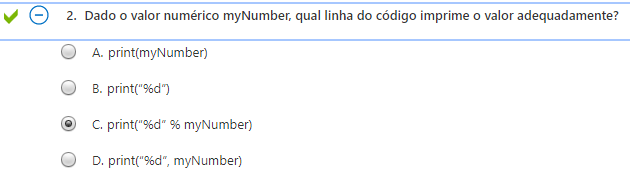
print("Voce irá receber {0:.2f}".format(chequeInt))

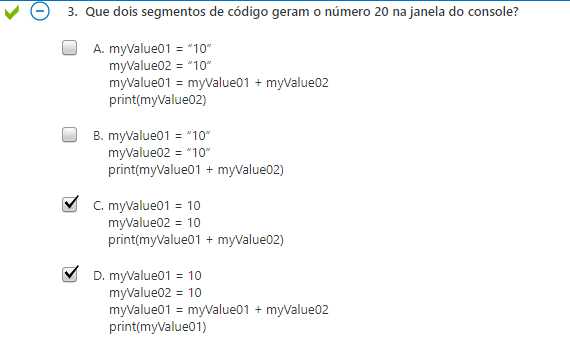
**Opções para conversão de variáveis.**

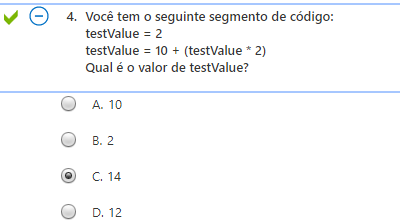


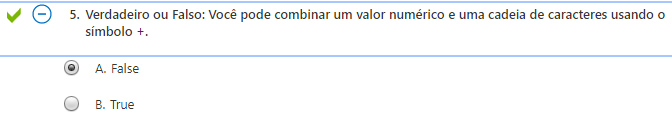
**Execícios:**





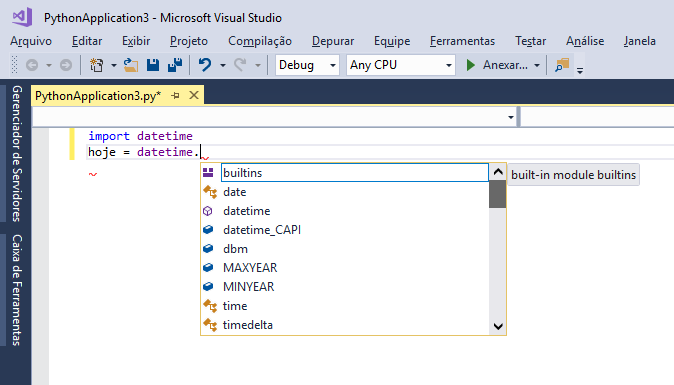






# Como trabalhar com datas e horas.

Este bloco tem como objetivo abordar datas, sempre na programação haverá a necessidade de controlar prazos, históricos, agendamentos, etc... Então para esta finalidade serão aplicados informações ou valores de data e hora. Então para atender este objetivo iremos abordar recursos adicionais, mas precisamente a biblioteca datetime, que irão disponibilizar recursos ou funções que agregam recursos ao Python.



Eventualmente teremos a necessidade de representar data de forma escrita ao invés de representações por números. Para isso será necessário usar o recurso **strftime** assim será possível especificar o formato da representação.

# Aula 05 - Trabalhando com datas

import datetime

dataAtual = datetime.date.today()

print(dataAtual)

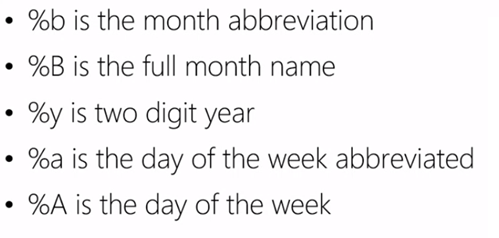
print(dataAtual.year)

print(dataAtual.month)

print(dataAtual.day)

print(dataAtual.strftime("%d %b, %Y"))

Para mais informações acesse [www.strftime.org](http://www.strftime.org)



Uma outra informação importante é que podemos informar na programação os parâmetros de localização, assim as configurações de idiomas e datas serão absorvidas no código de programação. Acesse <http://babel.pocoo.org/en/latest/>

**strptime()** pega um vetor de caractere e faz um objeto de data e hora.

Isso é entrada.

**strftime()** pega um objeto de data e hora e faz um vetor de caractere.

Essa é a saída e normalmente é chamada via format () ou print ().

Um recurso muito interessante é a possibilidade de comentar um texto grande, assim selecionando o texto e usando Ctrl+K em seguida Ctrl+C, todo o texto selecionado será comentado. Ctrl+K e Ctrl+U remove o comentário do texto.

Trabalhando com tempo.

# Trabalhando com tempo

import datetime

currentTime = datetime.datetime.now()

print(currentTime)

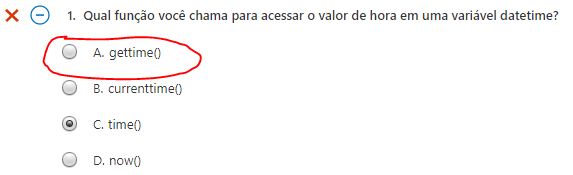
print(currentTime.hour)

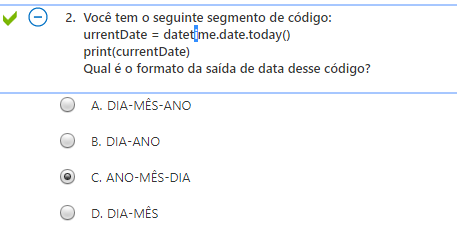
print(currentTime.minute)

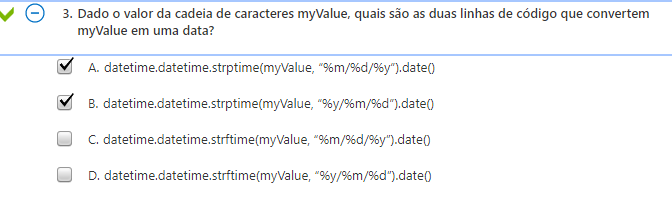
print(currentTime.second)

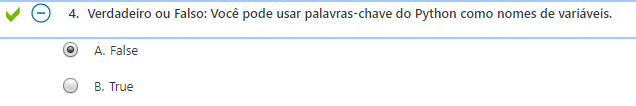
<https://dateutil.readthedocs.io/en/stable/>

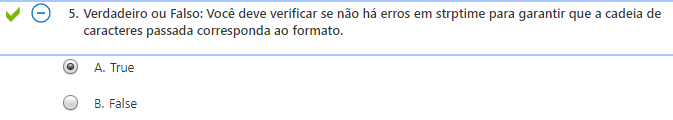
Exercícios:







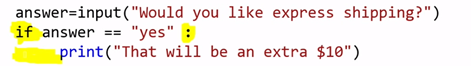




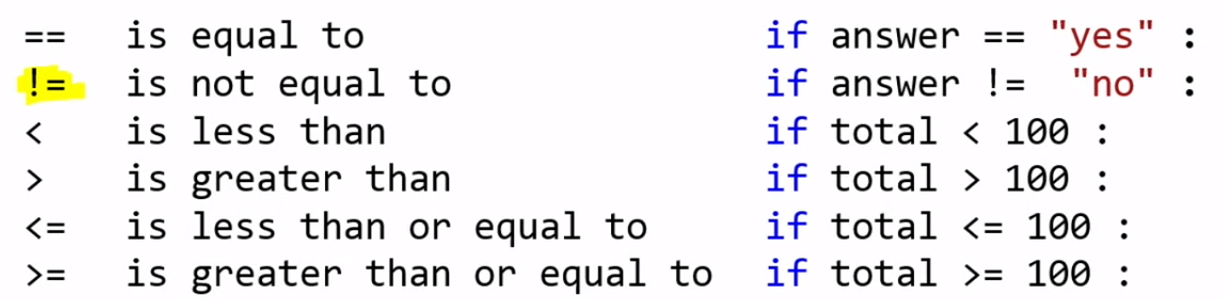
# Tomando decisões com o código.

A programação representa necessidades do mundo real, sua finalidade é desenvolver soluções para estes problemas. Então o processo decisório também deve ser representado e aplicado na programação.

Vamos analisar uma sintaxe simples.



**Operadores condicionais**



Ao avançarmos na programação iremos identificar que se a clausula if for atendida ou não devemos dar alguma resposta, segue exemplo.



# Tomando decisões com codigo

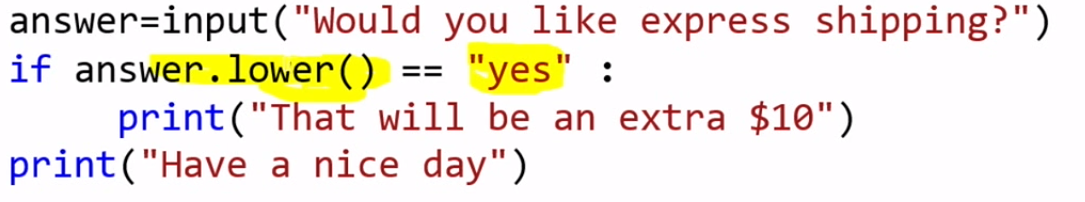
timeFavorito = input("Qual seu time? ")

if timeFavorito == "Flamengo" or timeFavorito =="flamengo":

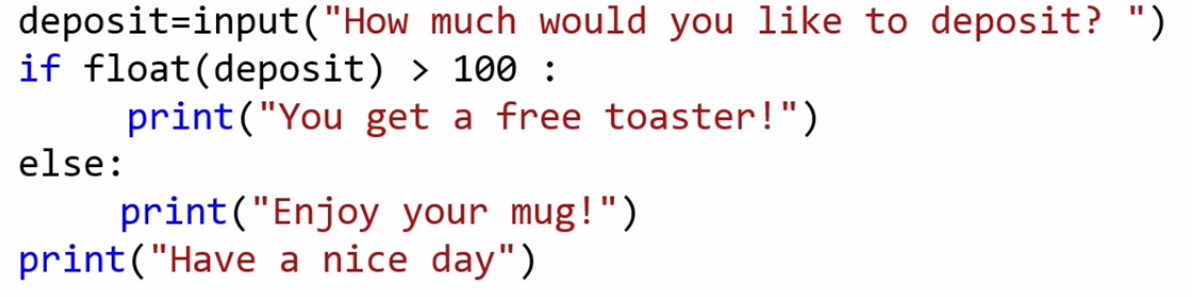
print("Parabêns seu time é top!")

print("Boa tarde!")

No exemplo a seguir irei representar um erro recorrente, pois podemos colocar o valor da variável de entrada como caixa alta ou baixa. Mas se o valor de comparação não acompanhar o valor de origem o código não vai funcionar.



Chegamos ao ponto em que se o if não for atendido devemos tomar outra ação, para isso use else. Segue sintaxe.



Também podemos usar variáveis boleanas no processo de decisão, assim quando fizermos referência da variável em vários pontos do código será mais fácil de gerenciar.

# Usando variaveis boleanas para tomar decisões

deposito = 100

if deposito > 500:

brinde = True

else:

brinde = False

if brinde :

print("Parabens, você ganhou um brinde! ")

print("Tenha um bom dia! ")

**Opção 2**

# Usando variaveis boleanas para tomar decisões

deposito = 600

brinde = False

if deposito > 500:

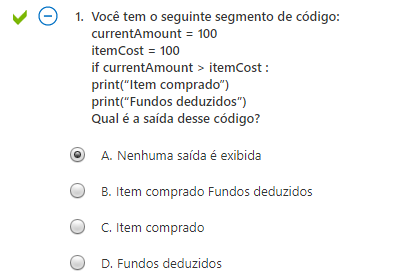
brinde = True

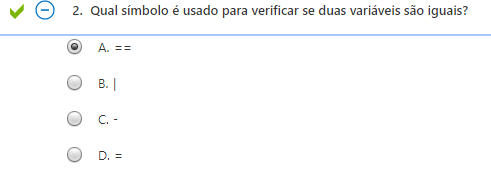
if brinde :

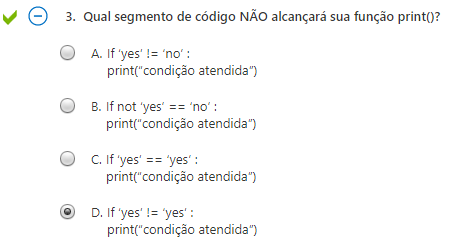
print("Parabens, você ganhou um brinde! ")

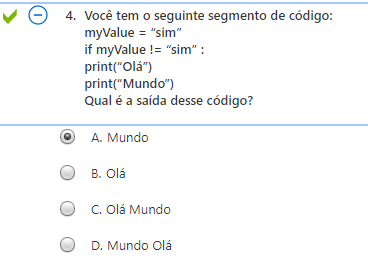
print("Tenha um bom dia! ")

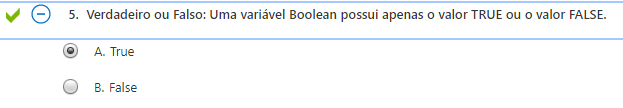
**Exercícios:**











# Decisões complexas.

Este bloco tem como objetivo abordar o avanço do bloco anterior, assim apresentando na programação como as decisões mais complexas devem ser aplicadas. Para isso iremos usar **if** e **elif**, quando qualquer uma das possibilidades forem atendidas a execução irá saltar para um item posterior a esta chamada **if, elif.**

# Decisões complexas.

# Vamos abordar um exemplo de possibilidades de respostas dependendo do idioma de entrada.

idioma = input("Qual idioma o Sr.(a) fala? ").upper()

if idioma == "PORTUGUES":

print("Bom dia ")

elif idioma == "INGLES":

print("Good Morning")

elif idioma == "ESPANHOL":

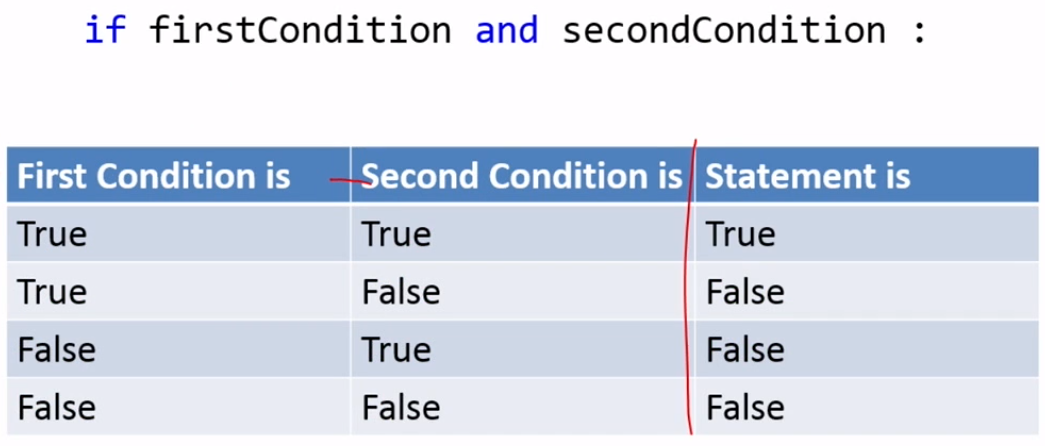
print("Buenos dias")

O código representado acima é funcional, desde que possua uma entrada controlada, como um painel de seleção que o usuário irá somente selecionar um dos três idiomas informados. Caso seja informado alguma resposta fora das opções a programação não irá funcionar. Para solucionar esta questão use **else**.

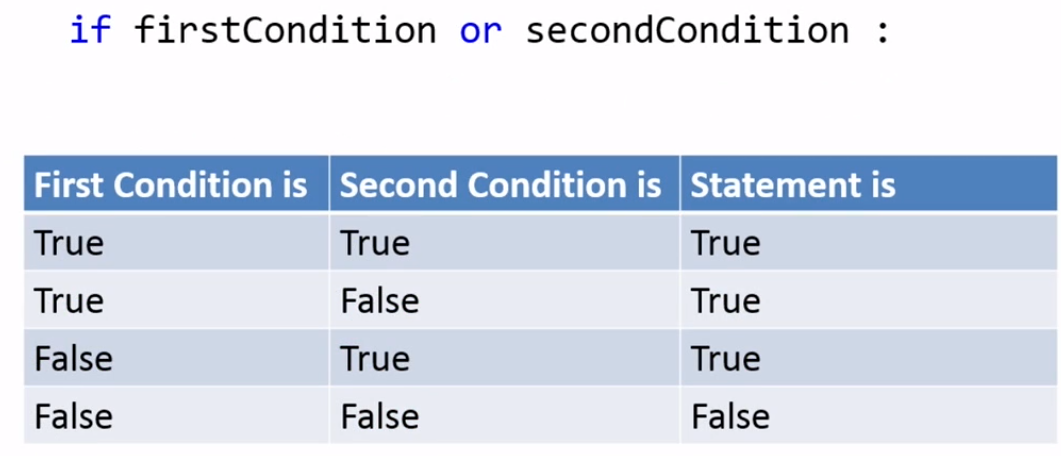
else:

print("Idioma desconhecido")

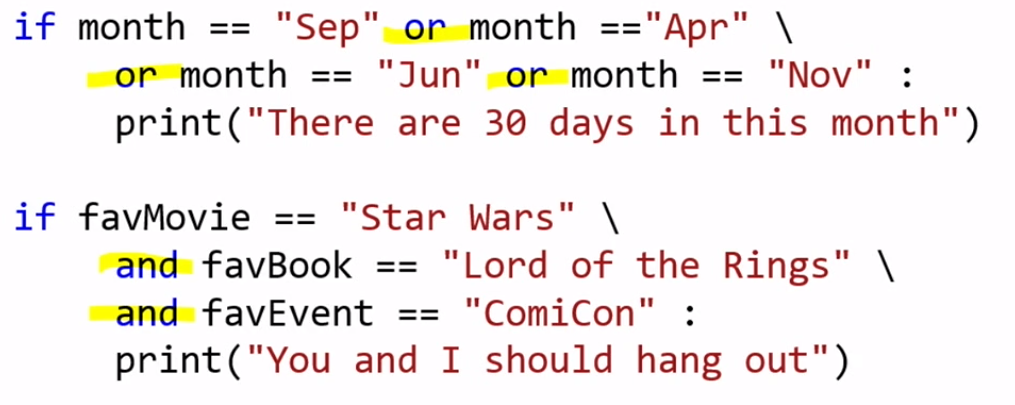
Também iremos nos deparar com processos decisórios onde duas ou mais condições devem ser atendidas, então para isso use **and**. Salvo que se uma das condições não forem atendidas este bloco de código não será executado.



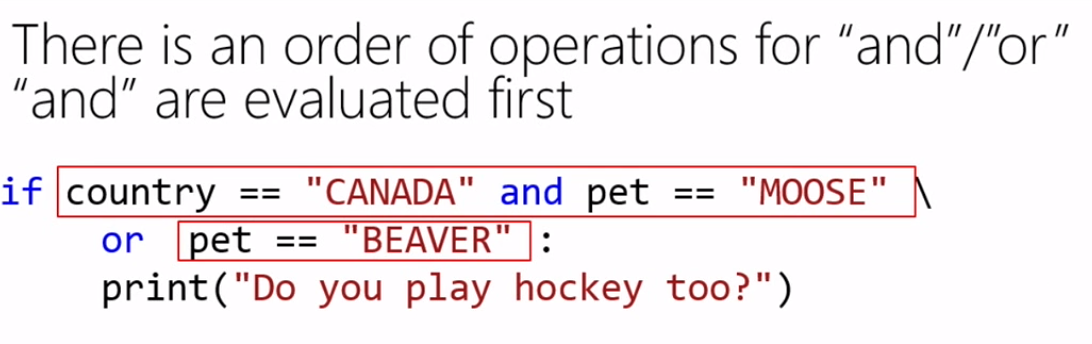
Assim como **and** foi aplicado também podemos aplicar **or**, mas o funcionamento é diferente. Caso uma das condições seja atendida o código será executado. Só não ira funcionar se as duas condições forem falsas.



Aplicação



Também é possível aplicar em conjunto **and** e **or**.



Mas a forma representada acima irá apresentar erro, pois irá retornar verdadeiro caso um dos pets sejam verdadeiros, independente da primeira validação country. Como solucionar isto:

# Validando and e or na mesma instrução, para isso iremos representar um processo de brindes de cartão de credito

# Se o cartão for outro ou prata e o consumo superior a valor X, o titular do cartão ganhará um brinde.

tipoCartao = input("Informe o tipo do cartão: ").upper()

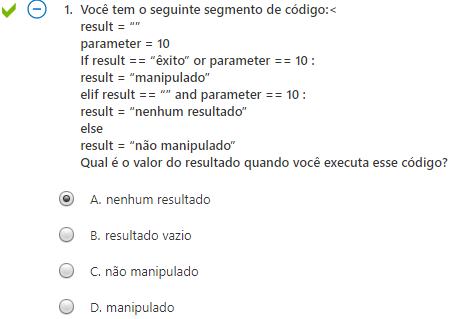
fatura = 1300.00

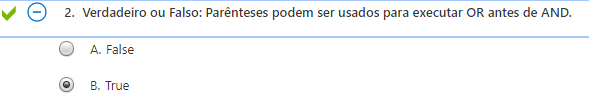
if fatura >= 1100.00 and (tipoCartao == "OURO" or tipoCartao == "PRATA"):

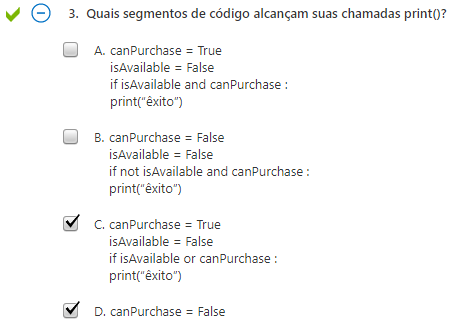
print("Parabêns! Você ganhou um brinde")

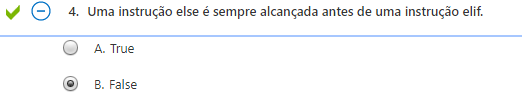
print("Tenha um bom dia!")

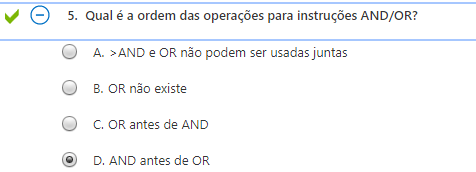
**Exercícios:**











# Repetição de eventos.

Para facilitar o aprendizado iremos utilizar a biblioteca turtle, que serve para efetuarmos desenhos baseados em direções e cores definidas no código.

# Usando a biblioteca turtle podemos representar um desenho como processo

import turtle

turtle.forward(100)

turtle.right(90)

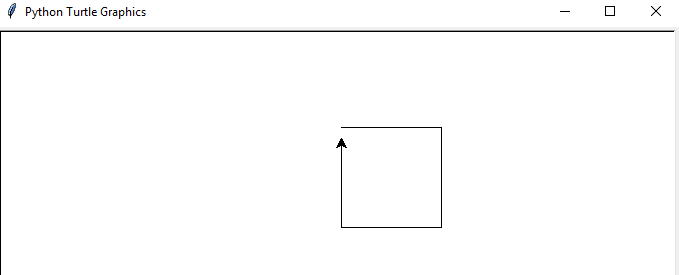
turtle.forward(100)

turtle.right(90)

turtle.forward(100)

turtle.right(90)

turtle.forward(100)



# Usando a biblioteca turtle com loop para desenha o mesmo quadrado

import turtle

for steps in range(4):

turtle.forward(100)

turtle.right(90)

# Também podemos colocar um loop dentro de outro loop

import turtle

for steps in range(4):

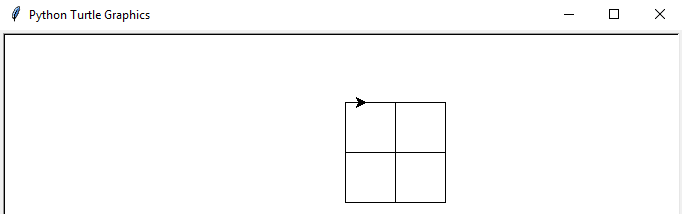
turtle.forward(100)

turtle.right(90)

for steps in range(4):

turtle.forward(50)

turtle.right(90)



# Tambem é possivel usar a biblioteca para realizar desenho mais elaborados

import turtle

for steps in range(5):

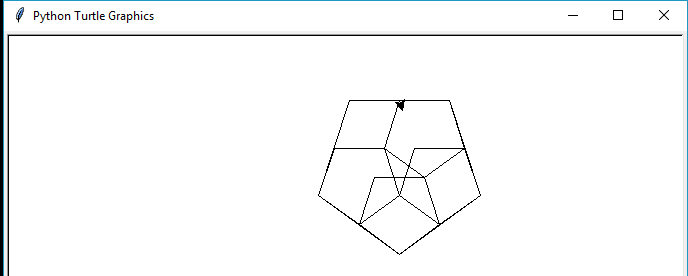
turtle.forward(100)

turtle.right(360/5)

for steps in range(5):

turtle.forward(50)

turtle.right(360/5)



# Uma boa prática de programação é passar os parâmetros em variáveis, assim reduzirá a mão de obra numa manutenção futura

import turtle

nr\_lados = 4

for steps in range(nr\_lados):

turtle.forward(100)

turtle.right(90)

for steps2 in range(nr\_lados):

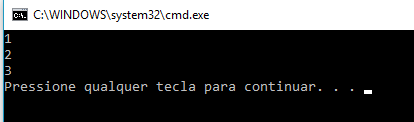
turtle.forward(50)

turtle.right(90)

# Podemos tambem definir um início e fim para o range de execução.

for steps in range(1,4):

print(steps)

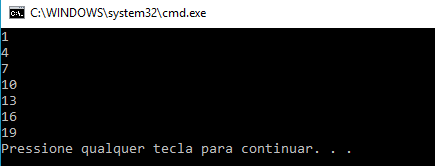


A contagem será iniciada em 1 mas não alcançará o 4, pois usa o conceito de até 4 no caso é a variável limite.

# Podemos tambem definir um salto dentro do início e fim do range. O salto é 3

for steps in range(1,20,3):

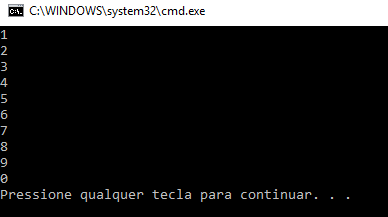
print(steps)



# Também há a necessidade de trabalharmos com arrays de variáveis. Salvo que serão sempre strings - Não carrega a informação range

for steps in [1,2,3,4,5,6,7,8,9,0]:

print(steps)



# Vamos representar uma execução de loop com usando a informação do array

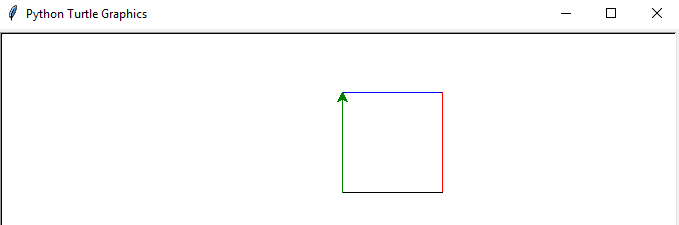
import turtle

for cores in ["blue","red","black","green"]:

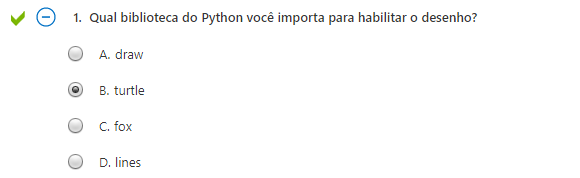
turtle.color(cores)

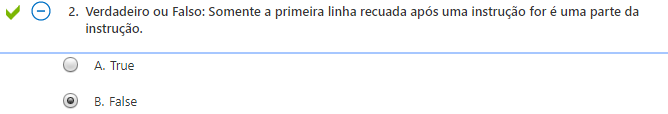
turtle.forward(100)

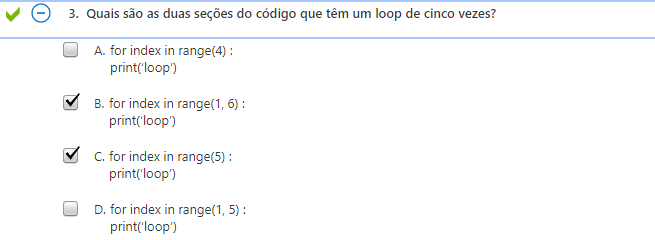
turtle.right(90)

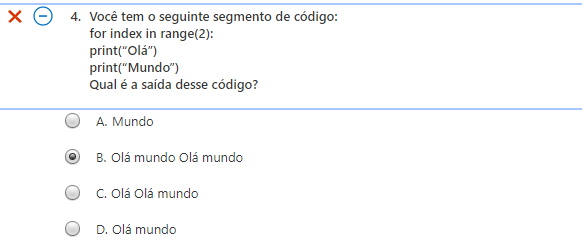


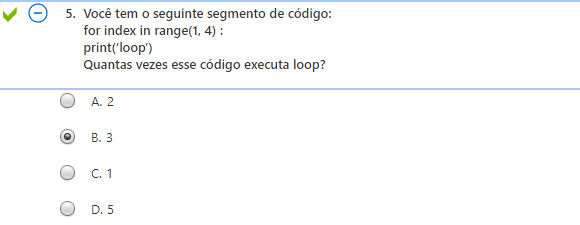
Exercícios:











# Repetição de eventos até a conclusão.

No modulo anterior efetuamos a programação usando for, mas o for funciona usando um range ou um array de execução. Agora iremos abordar a execução do programa até que uma demanda seja atendida. Ou ate mesmo uma execução constante, no aguardo de uma entrada.

Exemplos: Lavar louça até o fim, servir um café ao cliente que adentrar o estabelecimento, coleta contínua de dados de sensores...

Então o programa irá funcionar da seguinte maneira:

# while = Enquanto, então até que a resposta não seja atendida o sistema irá executar...

resposta = 0

while resposta != "12":

resposta = input("Quanto é 28 - 16 : ")

print("Você errou")

print("Parabéns, resposta correta ")

# Vamos misturar o while com turtle, para analisarmos o funcionamento

import turtle

contagem = 0

while contagem < 4:

turtle.forward(100)

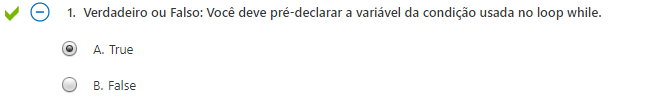
turtle.left(90)

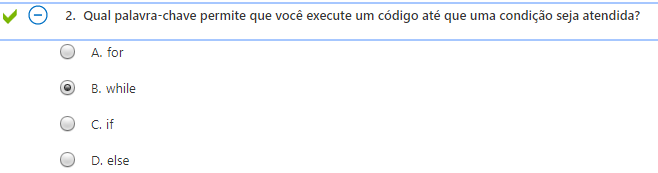
contagem = contagem + 1

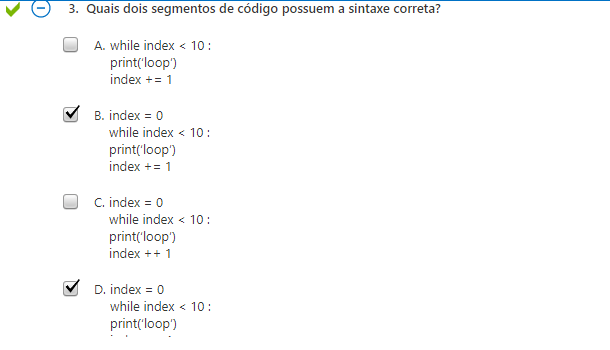
# O incremento também pode ser executado assim

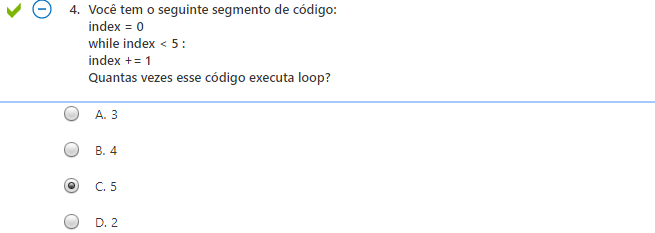
contagem += 1

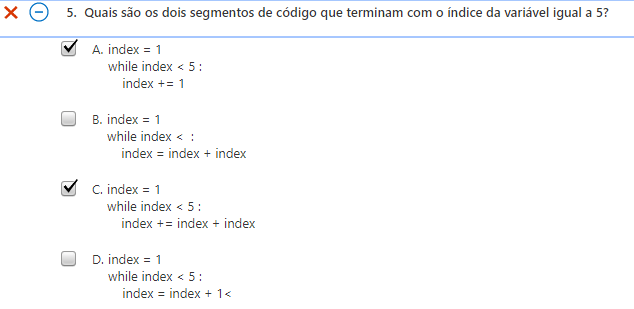
Exercícios:



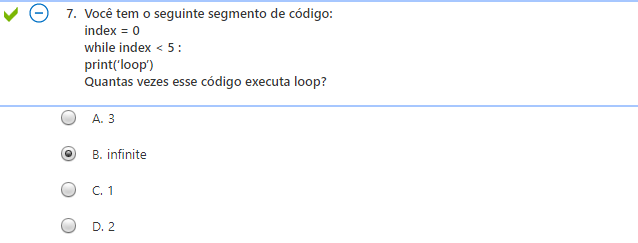












# Lista de lembranças.

A representação de lista é feita através de uma variável que recebe uma lista de itens entre colchetes e separados por vírgula. Diferentes tipos de dados podem ser armazenados, datas, strings, números, etc... Mas é extremamente recomendado que não misture itens em uma lista, como nome e telefone, veículo e placa, etc... É recomendado que se crie duas listas uma para o nome e outra para o telefone, uma para o veículo e outra para placa. Será mais fácil de gerenciar do que uma única lista com nome, telefone, nome, telefone...

Exemplos:

# Listas

alunos = []

alunos = [ "João Gabriel", "Daniel", "Bruna", "Renata", "Paulo"]

print(alunos)

numerosLoteria = []

numerosLoteria = ["02", "55", "43", "23", "08", "37"]

print(numerosLoteria)

# Navegando nos itens da lista, atenção o primeiro item é 0

print(alunos[0])

print(numerosLoteria[0])

print(alunos[4])

print(numerosLoteria[4])

# Um recurso legal é que também conseguimos imprimir a lista de trás pra frente

print(alunos[-1])

print(numerosLoteria[-1])

print(alunos[-4])

print(numerosLoteria[-4])

# Atualizando valores na lista.

alunos[0] = "Herick"

print(alunos)

# Remover valores na lista.

alunos.remove("João Gabriel")

print(alunos)

# Deletar valores na lista, um item específico.

del alunos[0]

print(alunos)

# Efetuar inclusão de aluno

alunos.append("Herick")

print(alunos)

# Trabalhando com index, o retorno será o primeiro item que encontrar

alunos = []

alunos = [ "João Gabriel", "Daniel", "Bruna", "Bruna", "Renata", "Paulo"]

print(alunos.index("Bruna"))

# Trabalhanddo com index, pesquisar um indice que não existe

alunos = []

alunos = [ "João Gabriel", "Daniel", "Bruna", "Bruna", "Renata", "Paulo"]

print(alunos.index("Herick"))

#ValueError: 'Herick' is not in list

# Então se quisermos fazer uma impressão colunar de todos registros da lista podemos efetuar um loop

for aluno in alunos:

print(aluno)

# Então ao efetuarmos a impressão com for é como fazer print(alunos[N]) N = se refere a cada um dos registros existentes

# Uma outra opção de fazer a contagem e impressão desta lista é usar len para definir o range de atuação do loop

for indice in range(len(alunos)):

print(alunos[indice])

print("Esta listagem possui {} aluno(s)".format(len(alunos)))

# Ou

qtdAlunos = 0

qtdAlunos = (len(alunos))

for indice in range(qtdAlunos):

print(alunos[indice])

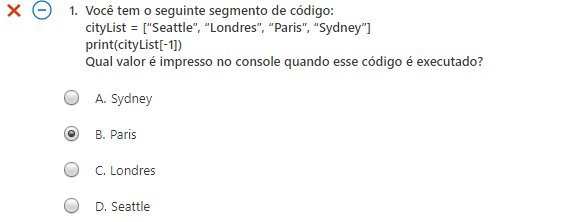
print("Esta lista possui {} aluno(s)".format(qtdAlunos))

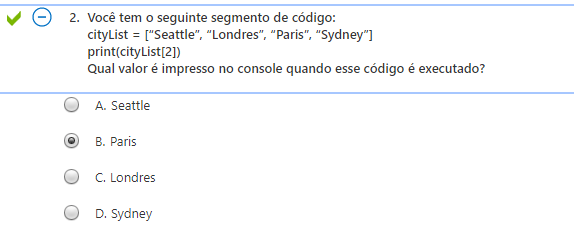
# Colocando a lista em ordem alfabética

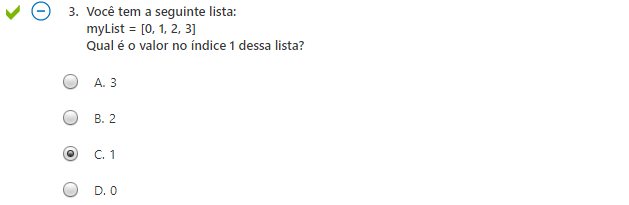
alunos.sort()

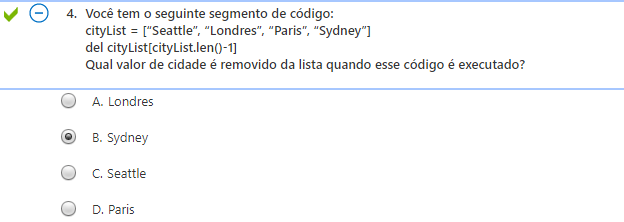
print(alunos)

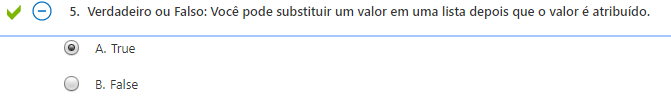
Exercícios:











# Como salvar informações em arquivos.

**Leitura de arquivos**

Avançando no nosso aprendizado continuamos com o foco na resolução de problemas do dia a dia e eventualmente iremos nos depara com situação em que precisaremos ler ou salvar dados em arquivos. Esta é uma questão que ocorre com recorrência, pois até os programas mais simples possuem arquivos de configuração ou parametrização que são consultados constantemente.

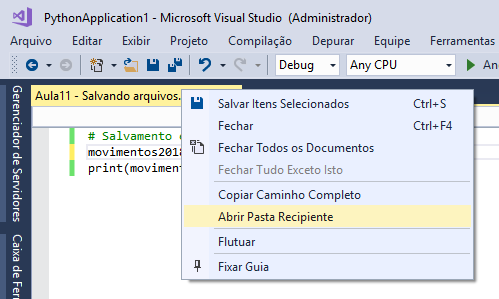
Exemplos: Um jogo lembra em que fase você parou, um livro eletrônico que salva seu ponto de leitura, arquivo de configuração do seu navegador web que salva suas preferencias e dados de navegação, etc...

Além dos exemplos citados há alguns anos atrás o mundo digital passou adotar algumas políticas de dados abertos, então assim é possível criar aplicativos para consumir dados sobre a última prova do ENEM, horários e trajetos de ônibus, trafego aéreo, informações do IBGE, dados econômicos, etc...

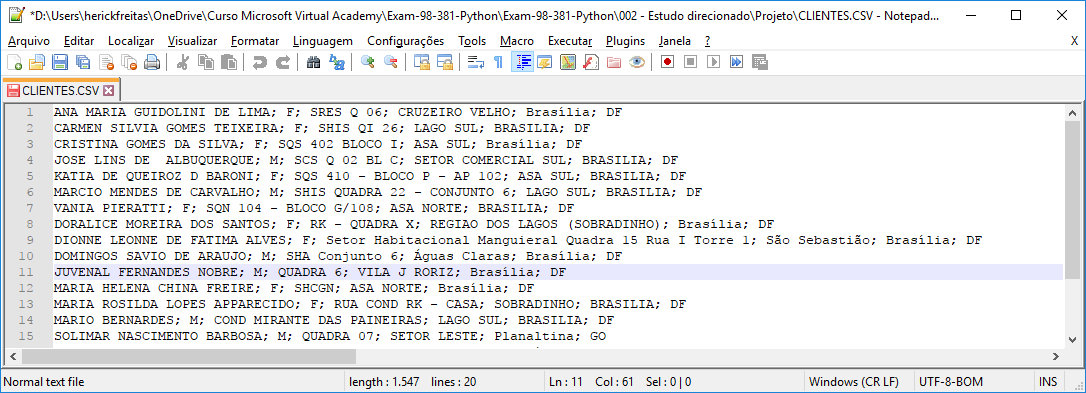
<http://www.data.rio/>

<http://dados.gov.br/>

Coloque o arquivo quer será usado dentro do diretório de recipiente.



Para o teste criei um arquivo com 20 registros contendo nome, sexo e endereço.



# Efetuando a leitura do arquivo

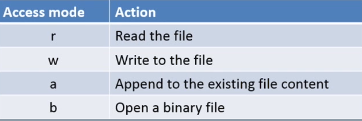
clientes = open("CLIENTES.csv", "r", encoding="utf8")

clientesModelo = clientes.read()

print(clientesModelo)



Modos de acesso



A opção **r** será usada para leitura de arquivos.

A opção **w** será usada para gravação de arquivos.

A opção **a** será usada para efetuar escrita em arquivos existentes.

A opção **b** será usada para abertura de arquivos binários.

# Efetuando leitura de linha

clientes = open("CLIENTES.csv", "r", encoding="utf8")

clientesModeloLinha = clientes.readline()

print(clientesModeloLinha)

# Efetuando leitura de CSV usando a biblioteca import CSV e transformando em listas, efetuando leitura linha a linha

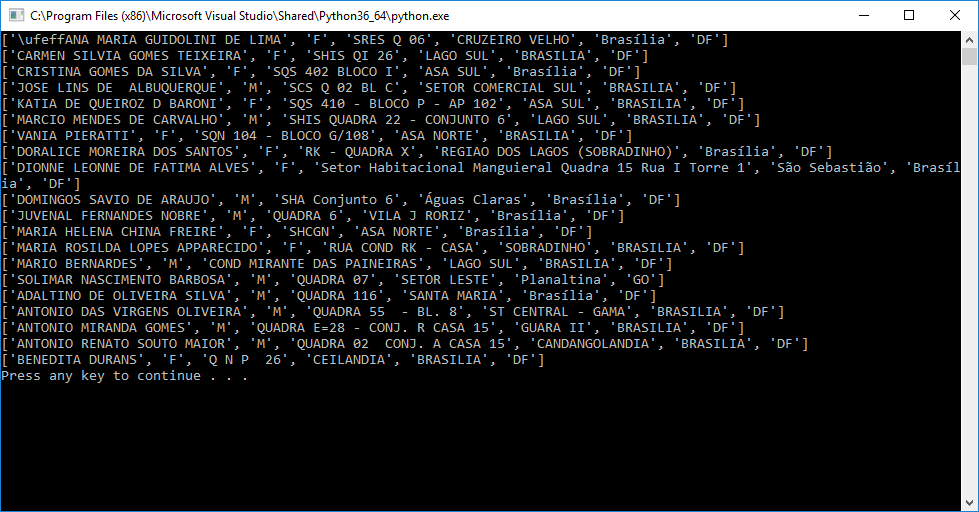
import csv

with open("CLIENTES.csv", "r", encoding="utf8") as myCSVFile:

clientes = csv.reader(myCSVFile, delimiter=";")

for linha in clientes:

print(linha)



# Efetuando leitura do item da linha

import csv

with open("CLIENTES.CSV", "r", encoding="utf8") as myCSVFile:

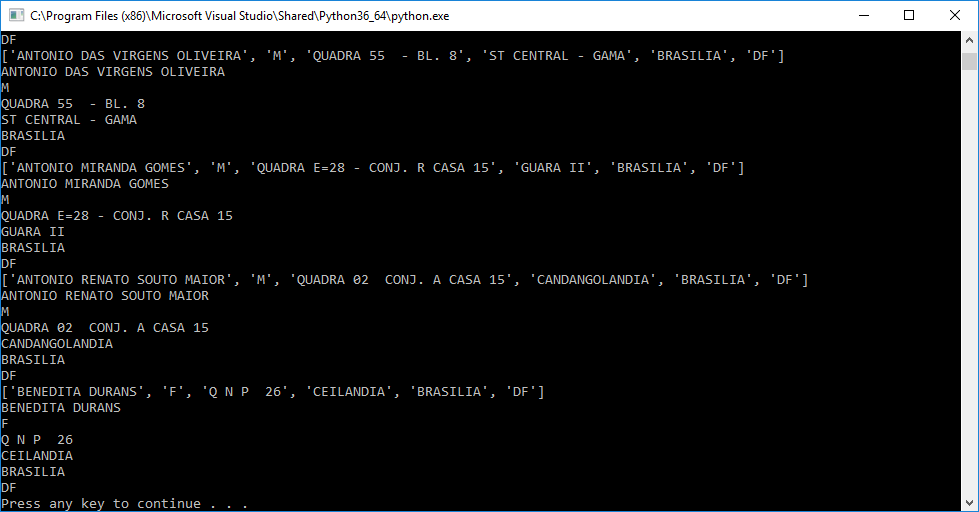
clientes = csv.reader(myCSVFile, delimiter=";")

for linha in clientes:

print(linha)

for item in linha:

print(item)



# Alterando separador do arquivo para vizualização

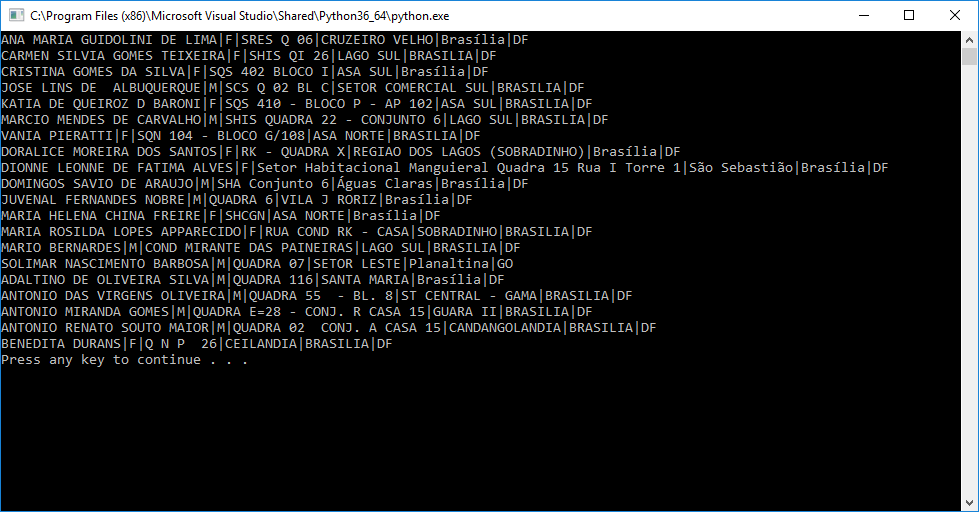
import csv

with open("CLIENTES.CSV", "r", encoding="utf8") as myCSVFile:

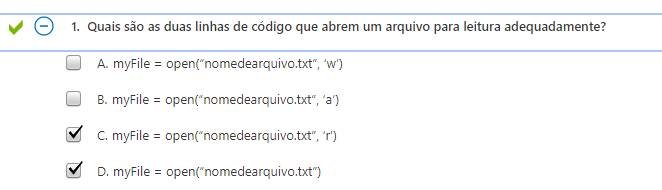
clientes = csv.reader(myCSVFile, delimiter=";")

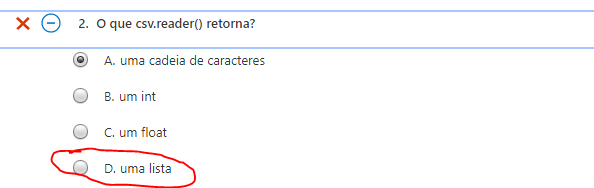
for linha in clientes:

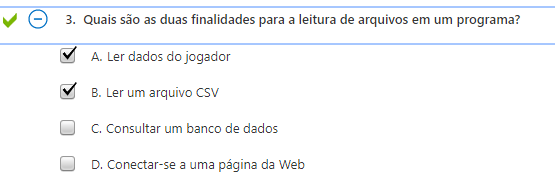
print('|'.join(linha))

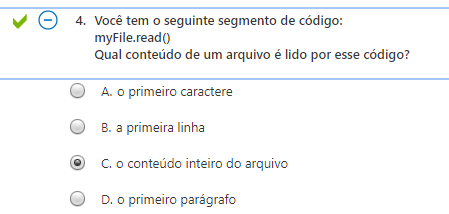


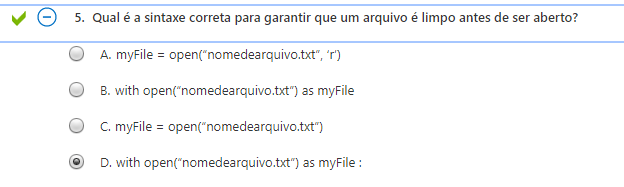
**Exercícios:**







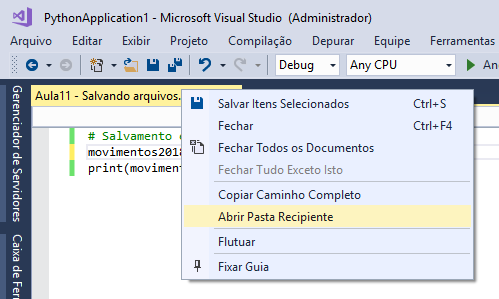




# Leitura a parti de arquivos

**Leitura de arquivos**

Quando referenciar o arquivo para gravação é indispensável informar a extensão (CSV, TXT, etc...) do mesmo. O recurso de abrir pasta recipiente também se aplica a gravação de arquivos.



############## WRITE

# Efetuando a leitura de um arquivo, mesmo se tratando de um gravação o comando continua sendo open.

# Usando w - Caso o arquivo não exista ele será criado na execução, se existir o mesmo será substituído

# Usando a - Caso o arquivo não exista ele será criado na execução, se existir o os dados gerados serão incluídos.

# Caso ocorra a necessidade de efetuar a leitura e gravação no mesmo código use w+

demo = open("demo.txt","w")

demo.write("Criando primeira linha \n")

demo.write("Criando segunda linha \n")

demo.write("Criando terceira linha \n")

demo.write("Welcome!!")

## Atenção é indispensável o uso do close, senão o arquivo poderá abresentar dado sujo ou inválido... por estar em uso.

demo.close()

#print("A gravação ocorreu com sucesso")

# Efetuando uma interação para gravar em arquivo

dados = input("Informe os nomes dos convidados: ")

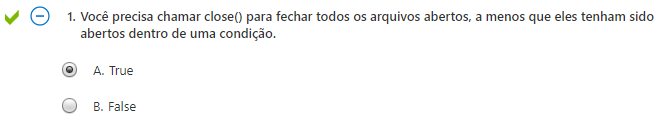
arquivo = open("demo.csv","w")

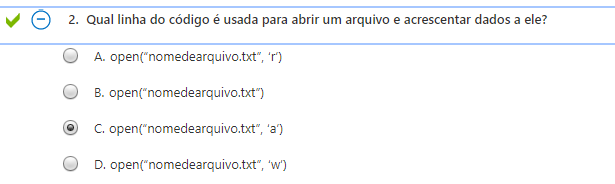
arquivo.write(dados)

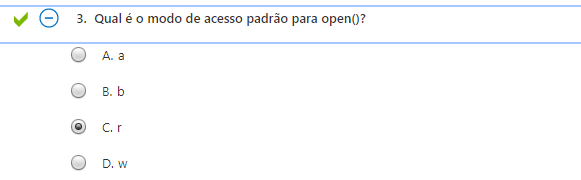
arquivo.close()

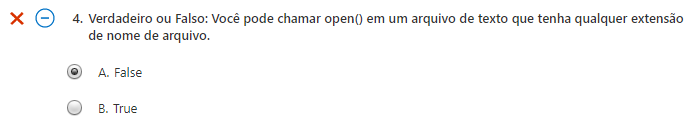
print("Dados gravados com sucesso! ")

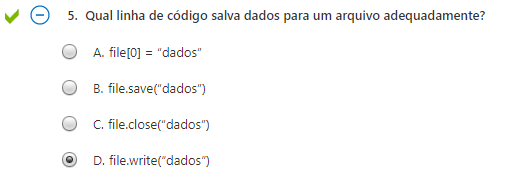
**Exercícios:**











# Funções.

Funções são aplicadas para que o mesmo programa possa ser usado repetidas vezes. Senão seria necessário copiar e colar o programa inteiro várias vezes, assim quando houvesse a necessidade de qualquer ajuste ou alteração vários códigos precisariam ser ajustados.

**Introdução**: Uma função é uma seção de código de programação nomeada que será reutilizável.

Então uma função tem a finalidade de facilitar a programação, evitando repetição excessiva de códigos e minimizando o trabalho nos momentos de manutenção dos codigos. É muito importante nomear as funções com nomes que realmente façam referência aos seus recursos. Exemplos: escreverDadosArquivo ou coletarDadosUsuario

**Sintaxe básica**

# Sintaxe de uma Função

def printMenssagem():

print("Hello world!")

return

# Sintaxe de execução da função

printMenssagem()

**Utilizando função main()**

Aplicando este método múltiplas funções podem ser executadas em sequência, através de uma mesma chamada.

# Sintaxe de uma Função utilizando função main()

def main():

printMenssagem()

return

def printMenssagem():

print("Hello world!")

return

# chamada da função

main()

**Utilizando passagens de parâmetro na função**

# Definindo função com variável (texto)

def printMessangem(texto):

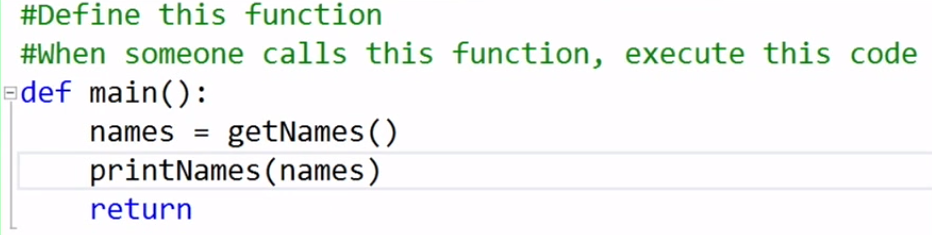
print(texto)

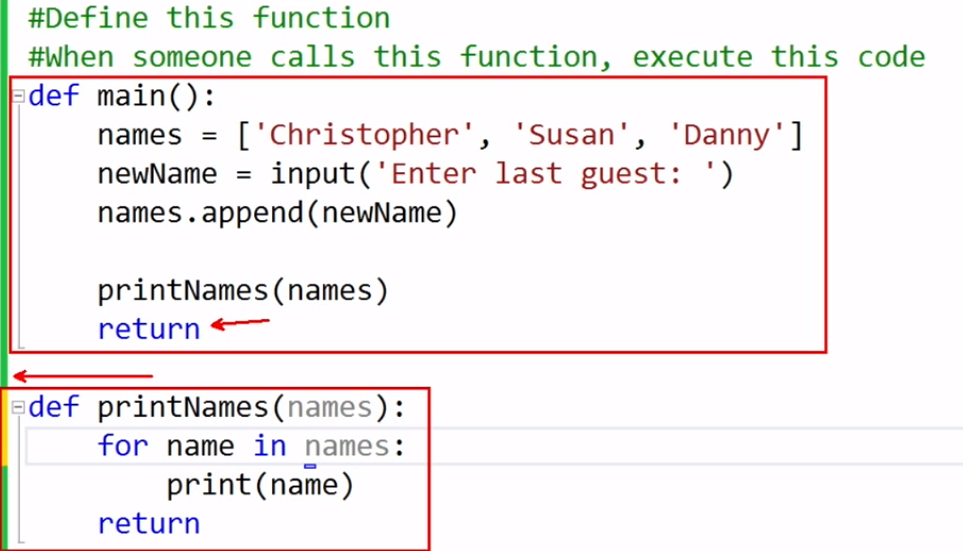
return

# Efetuando chamada da função e informando o texto para o parêmtro

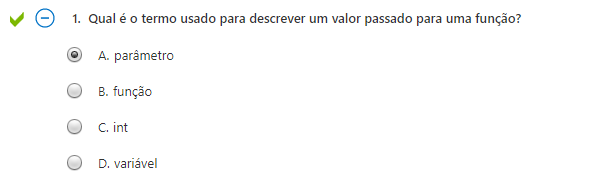
printMessangem("Hello World!")

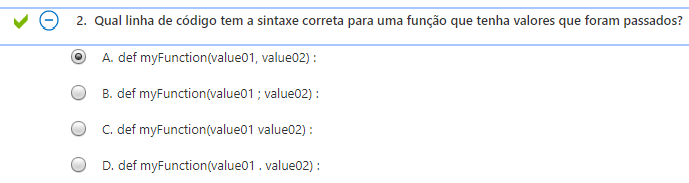
**Aumentando complexidade das funções**

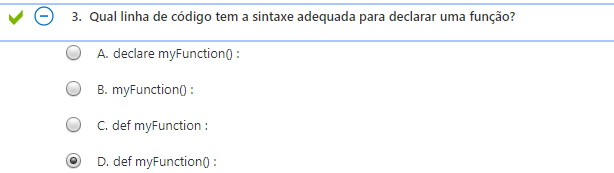


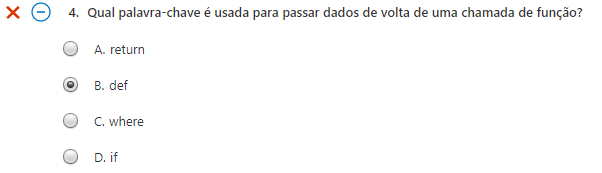


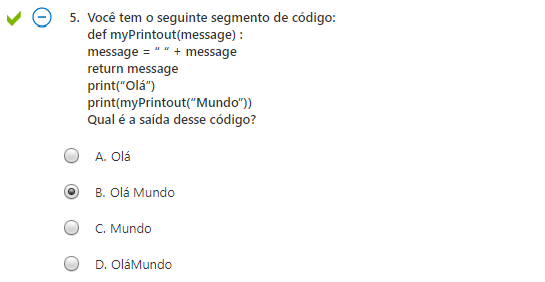
**Exercícios:**











# Manipulação de erros.

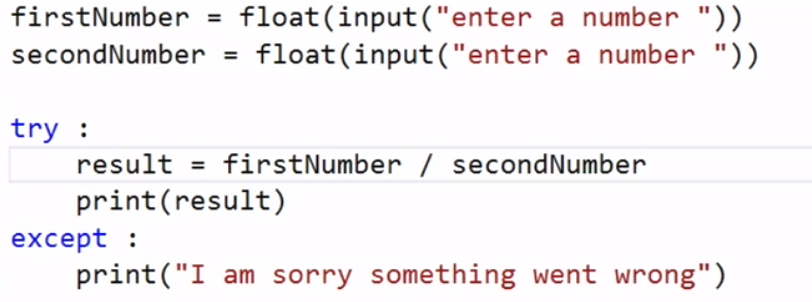
Mesmo que você seja muito atento e organizado as vezes as coisas irão sair errado, talvez alguma falha no código ou algum arquivo necessário para alimentar o código foi movido do local adequado. As vezes o usuário não informa os dados conforme o indicado como por exemplo yyyy/mm/dd ao invés de dd/mm/yyyy.

Quando erros ocorrerem o que devemos analisar?

- Primeiramente analise a sintaxe, utilize o depurador do Visual Studio para analisar a sintaxe do programa escrito. Atenção para aspas, parênteses, dois pontos, as linhas vermelhas sublinhadas serão as dicas de erros de sintaxe.

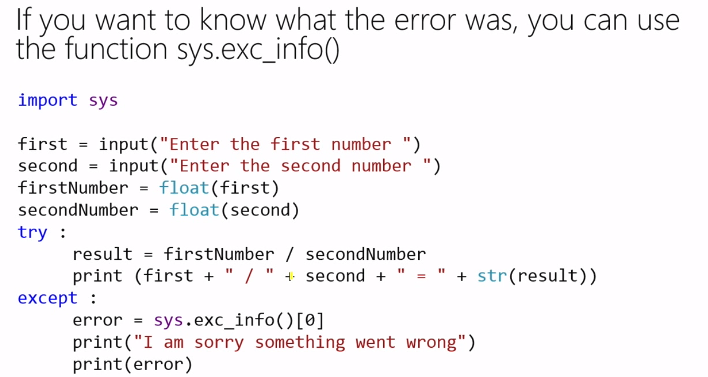
- Quando executar o código e uma mensagem de erro for apresentada, identifique o texto citado na mensagem. Certamente será informando onde está o erro e o motivo da crítica, as vezes até um modelo de sintaxe correto será apresentado.

Há uma abordagem para tratamentos de erros que podemos aplicar em programação, que é try: e except: onde podemos definir alguma ação caso ocorra falha na execução. Este recurso tem a finalidade de tratar as mensagens de erro com elegância, para que o usuário não tenha acesso a erros sistêmicos de difícil compreensão. Exemplo: divisão por 0.



A mensagem que será apresentada não irá dar qualquer referência sobre o erro ocorrido, então para deixar a mensagem mais objetiva vamos usar a biblioteca **sys** que irá disponibilizar **sys.exc\_info()[0]** que apresentará a detalhes sobre o erro ocorrido.

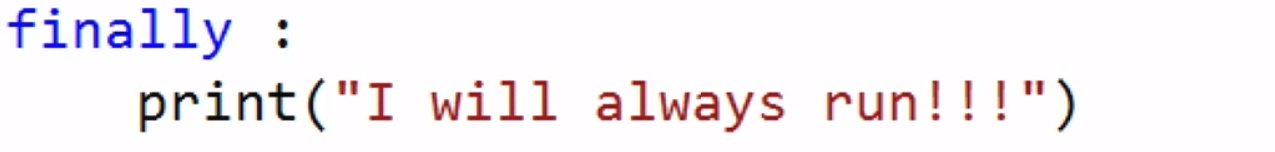
**Sintaxe**



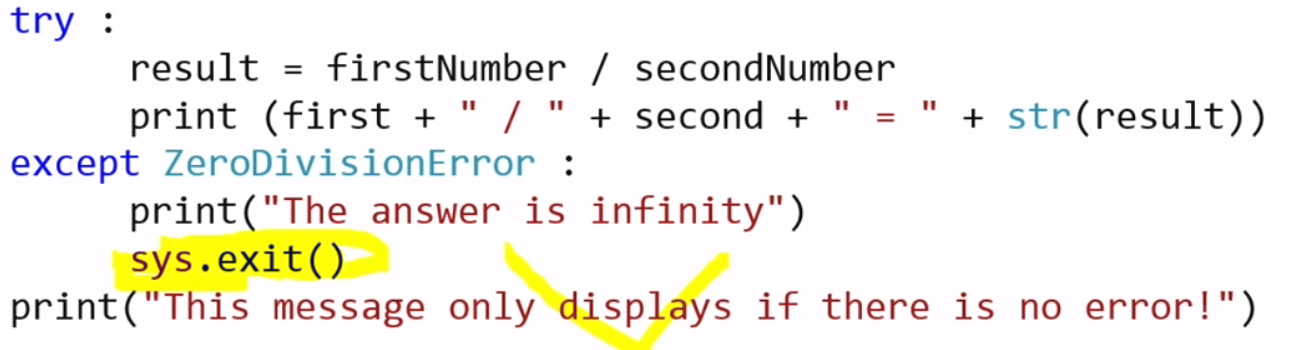
**Resultado**



Finally é um grupo de código que sempre será executado, deve ficar disposto ao final da sintaxe de programação após a except. Não é muito usado, pois o print direto funcionaria.

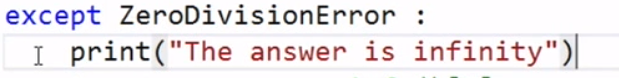


Atenção: Dependendo da sua programação, pode haver o caso a seguir. Onde devido ao exit o print a seguir não será executado. Mas se estivesse com o grupo finally: ele seria executado, independente do comando exit.

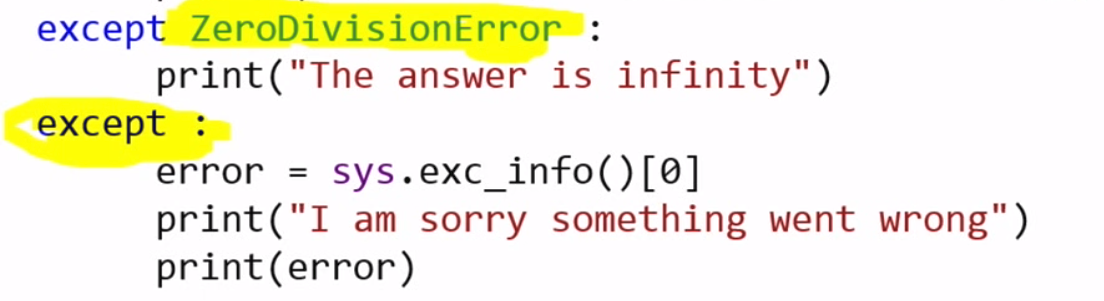


Uma abordagem muito importante é identificar possíveis erros antes que eles aconteçam, assim quando você escrever um programa que usa um arquivo em sua sintaxe, faço uma validação se o arquivo está no diretório padrão antes da execução do programa. Assim irá tratar possíveis problemas na origem.

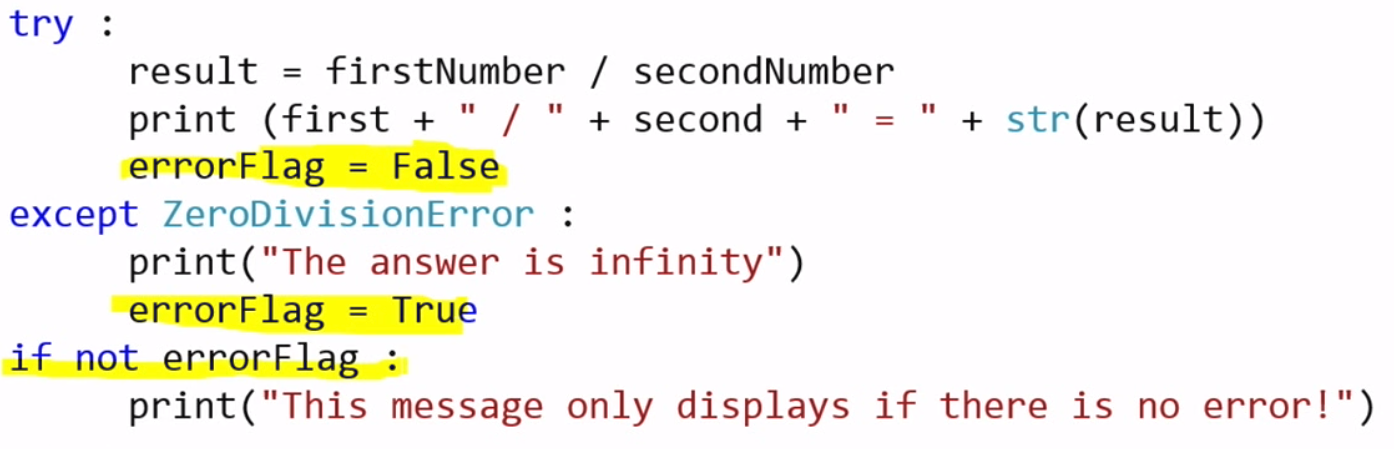
Também é possível citar o erro na exceção, assim dando uma resposta mais objetiva ao operador do programa.



Trabalhando com mais possibilidades.

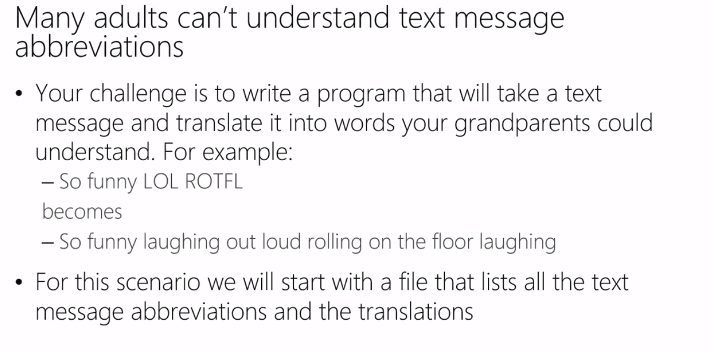


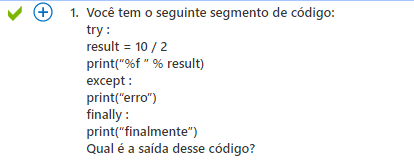
Uma outra abordagem seria trabalharmos com variáveis... segue exemplo:

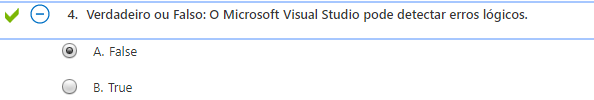


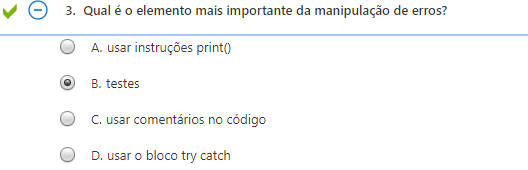
É muito importante criar bateria de testes para seus programas, informe strings ou floats no lugar de inteiros. Force erros, assim poderá criar mensagens de alertas e definir uma abordagem de programação mais objetiva. Faça coisas que você acredita que os usuários possam fazer.

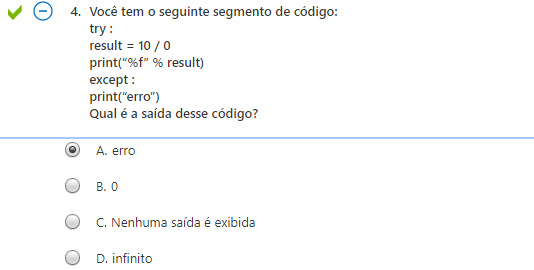
**Exercícios:**













# Extras

## **Módulos - Operando interfaces de sistema, diretórios**

The [os](https://docs.python.org/3/library/os.html#module-os) module provides dozens of functions for interacting with the operating system:

**>>> import** **os**

**>>>** os.getcwd() *# Return the current working directory*

'C:\\Python36'

**>>>** os.chdir('/server/accesslogs') *# Change current working directory*

**>>>** os.system('mkdir today') *# Run the command mkdir in the system shell*

0

Be sure to use the import os style instead of from os import \*. This will keep [os.open()](https://docs.python.org/3/library/os.html#os.open) from shadowing the built-in [open()](https://docs.python.org/3/library/functions.html#open) function which operates much differently.

The built-in [dir()](https://docs.python.org/3/library/functions.html#dir) and [help()](https://docs.python.org/3/library/functions.html#help) functions are useful as interactive aids for working with large modules like [os](https://docs.python.org/3/library/os.html#module-os):

**>>> import** **os**

**>>>** dir(os)

<returns a list of all module functions>

**>>>** help(os)

<returns an extensive manual page created from the module's docstrings>

For daily file and directory management tasks, the [shutil](https://docs.python.org/3/library/shutil.html#module-shutil) module provides a higher level interface that is easier to use:

**>>> import** **shutil**

**>>>** shutil.copyfile('data.db', 'archive.db')

'archive.db'

**>>>** shutil.move('/build/executables', 'installdir')

'installdir'

## **Módulos - Operando interfaces de sistema, arquivos**

Common utility scripts often need to process command line arguments. These arguments are stored in the [sys](https://docs.python.org/3/library/sys.html#module-sys) module’s argv attribute as a list. For instance the following output results from running python demo.py one two three at the command line:

**>>> import** **sys**

**>>>** print(sys.argv)

['demo.py', 'one', 'two', 'three']

The [getopt](https://docs.python.org/3/library/getopt.html#module-getopt) module processes sys.argv using the conventions of the Unix [getopt()](https://docs.python.org/3/library/getopt.html#module-getopt) function. More powerful and flexible command line processing is provided by the [argparse](https://docs.python.org/3/library/argparse.html#module-argparse)module.

## **Módulos – Saídas de erros e programando conclusão.**

The [sys](https://docs.python.org/3/library/sys.html#module-sys) module also has attributes for stdin, stdout, and stderr. The latter is useful for emitting warnings and error messages to make them visible even when stdout has been redirected:

>>>

**>>>** sys.stderr.write('Warning, log file not found starting a new one**\n**')

Warning, log file not found starting a new one

The most direct way to terminate a script is to use sys.exit().

## **Módulos – Aplicando funções matemáticas em strings**

The [re](https://docs.python.org/3/library/re.html#module-re) module provides regular expression tools for advanced string processing. For complex matching and manipulation, regular expressions offer succinct, optimized solutions:

**>>> import** **re**

**>>>** re.findall(r'\bf[a-z]\*', 'which foot or hand fell fastest')

['foot', 'fell', 'fastest']

**>>>** re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')

'cat in the hat'

When only simple capabilities are needed, string methods are preferred because they are easier to read and debug:

**>>>** 'tea for too'.replace('too', 'two')

'tea for two'

## **Módulos – Matemática**

The [math](https://docs.python.org/3/library/math.html#module-math) module gives access to the underlying C library functions for floating point math:

**>>> import** **math**

**>>>** math.cos(math.pi / 4)

0.70710678118654757

**>>>** math.log(1024, 2)

10.0

The [random](https://docs.python.org/3/library/random.html#module-random) module provides tools for making random selections:

**>>> import** **random**

**>>>** random.choice(['apple', 'pear', 'banana'])

'apple'

**>>>** random.sample(range(100), 10) *# sampling without replacement*

[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]

**>>>** random.random() *# random float*

0.17970987693706186

**>>>** random.randrange(6) *# random integer chosen from range(6)*

4

The SciPy project <[https://scipy.org](https://scipy.org/)> has many other modules for numerical computations.

## **Módulos – Datas e tempos**

The [datetime](https://docs.python.org/3/library/datetime.html#module-datetime) module supplies classes for manipulating dates and times in both simple and complex ways. While date and time arithmetic is supported, the focus of the implementation is on efficient member extraction for output formatting and manipulation. The module also supports objects that are timezone aware.

**>>>** *# dates are easily constructed and formatted*

**>>> from** **datetime** **import** date

**>>>** now = date.today()

**>>>** now

datetime.date(2003, 12, 2)

**>>>** now.strftime("%m-*%d*-%y. *%d* %b %Y is a %A on the *%d* day of %B.")

'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

**>>>** *# dates support calendar arithmetic*

**>>>** birthday = date(1964, 7, 31)

**>>>** age = now - birthday

**>>>** age.days

14368

**Todos os objetos date, datetime e time suportam um método strftime (format), para criar uma string representando o tempo sob o controle de um formato explícito corda.**

**Aqui está uma lista dos códigos de formato com sua diretiva e significado.**

% o nome abreviado do dia da semana de um local.

% A nome completo do dia da semana de Locale.

% b nome abreviado do mês do local.

% B nome completo do mês do Locale.

% c Representação de data e hora apropriada do Locale.

% d Dia do mês como um número decimal [01,31].

% f Microssegundo como um número decimal [0,999999], preenchido com zero à esquerda

% H Hour (relógio de 24 horas) como um número decimal [00,23].

% I Hora (relógio de 12 horas) como um número decimal [01,12].

% j Dia do ano como um número decimal [001,366].

% m Mês como um número decimal [01,12].

% M Minuto como um número decimal [00,59].

% p Equivalente a Locale de AM ou PM.

% S Segundo como um número decimal [00,61].

% U Número da semana do ano (domingo como o primeiro dia da semana)

% w Dia da semana como um número decimal [0 (domingo), 6].

% W Número da semana do ano (segunda-feira como o primeiro dia da semana)

% x representação de data apropriada do Locale.

% X representação de tempo apropriada do Locale.

% y Ano sem século como um número decimal [00,99].

% Y Ano com o século como um número decimal.

% z offset UTC no formato + HHMM ou -HHMM.

% Z Nome do fuso horário (string vazia se o objeto for ingênuo).

%% Um caractere literal '%'.