-------------Capitulo 2 - Variaveis\_Tomada de decisão e laços de repetição--------------

**Tomadas de decisões**

Detalhe para as linhas 3 e 4 agora, onde estamos fazendo conversões de *string* para *int* (na linha 3) e de *string* para *float* (na linha 4). Isso porque o *input*() captura apenas *strings*, logo após capturado o valor, convertemos para o tipo de dado que desejamos armazenar em sua respectiva variável.

Para finalizar, vamos acrescentar mais quatro linhas no final do código e poderemos comprovar os tipos de dados que estão sendo utilizados pelas variáveis. Utilizaremos mais uma função, chamada ***type()***. Esta função retorna o tipo do dado do que estiver dentro dos seus parênteses, conforme observamos no código abaixo:

nome = input("Digite um funcionario: ")

empresa = input("Digite uma instituicao: ")

qtde\_funcionarios=int(input("Digite a qtde de funionarios"))

mediaMensalidade=float(input("Digite a media mensalidade"))

print(nome + "trabalha na empresa" + empresa)

print("Possui: ", qtde\_funcionarios, "funcionarios.")

print( "A media da mensalidade e de : "+ str(mediaMensalidade))

print("===============Verifique os tipos de dados abaixo: =================" )

print( "O tipo de dados da variaval [nome] é: ", type(nome))

print( "O tipo de dados da variaval [empresa] é: ", type(empresa))

print( "O tipo de dados da variaval [qtde\_funcionarios] é: ", type(qtde\_funcionarios))

print( "O tipo de dados da variaval [mediaMensalidade] é: ", type(mediaMensalidade))

**Decisões simples**

Nao eh aconselhado pois tem 3 variaveis, e variáveis consome recursos.

nome=input("Digite o nome: ")

idade=int(input("Digite a idade"))

prioridade="NAO"

if idade>=65:

prioridade="SIM"

print( "O paciente " + nome+ "possui atendimento prioritario?" + prioridade)

Agora economizamos recursos. Perceba que, no exemplo anterior, (DecisaoSimples), utilizamos três variáveis e, como já mencionado, variáveis ocupam espaço em memória.

nome=input("Digite o nome: ")

idade=int(input("Digite a idade"))

if idade>=65:

print("O paciente "+ nome + "tem atendimento prioritario!")

else:

print("O paciente "+ nome + "nao tem atendimento prioritario!")

**Decisões compostas**

Também poderemos encontrar uma situação em que o Verdadeiro e o Falso, simplesmente, não sejam suficientes. Por exemplo, vamos imaginar que pessoas com idade igual ou superior a 65 receberão atendimento prioritário, mas que também pessoas com suspeita de doenças infecto-contagiosas deverão ser direcionadas para uma sala de espera distinta:

upper () 🡪 deixar maisculo

= 🡪 atribuição

== 🡪 comparação

nome=input("Digite o nome: ")

idade=int(input("Digite a idade"))

doenca\_infect= input("Suspeita de doenca infecto-contagiosa?").upper()

if idade>=65:

print("O paciente "+ nome + "tem atendimento prioritario!")

elif doenca\_infect =="SIM":

print("O paciente " + nome + " precisa ir para uma sala reservada")

else:

print("O paciente "+ nome + " nao tem atendimento prioritario!")

**DECISAO ENCADEADA**

Primeiro vamos identificar a idade do paciente para definir se ele terá atendimento prioritário ou não, e depois se o paciente está ou não sob a situação suspeita de doença infecto-contagiosa. Vamos para o seguinte caso: mulheres grávidas também são consideradas para o atendimento prioritário (sala Branca ou Amarela). Você vai perguntar para todos os pacientes se eles estão grávidos? Não, apenas para as mulheres. Então, você perguntaria para todas as mulheres? Não, você não precisaria perguntar para as mulheres com idade igual ou superior a 65 anos, assim também poderia descartar crianças com menos de 10 anos.

nome=input("Digite o nome: ")

idade=int(input("Digite a idade"))

doenca\_infect= input("Suspeita de doenca infecto-contagiosa?").upper()

if idade>=65:

print("Paciente com prioridade")

if doenca\_infect=="SIM":

print("Encaminhe para a sala AMARELA")

elif doenca\_infect=="NAO":

print("Encaminhe para a sala Branca")

else:

print("Responda a suspeita de doenca infectocontagiosa com SIM ou NAO")

else:

print("Paciente sem prioridade")

if doenca\_infect=="SIM":

print("Encaminhe para a sala AMARELA")

elif doenca\_infect=="NAO":

print("Encaminhe para a sala Branca")

else:

print("Responda a suspeita de doenca infectocontagiosa com SIM ou NAO")

FOR

Repare que, para multiplicarmos os dois valores, utilizamos o operador “\*”. No range(), definimos que os valores gerados deverão estar entre 1 e 11, com incremento de 1 em 1. O “11” foi definido porque ele não se inclui no range, ou seja, quando chegar em 11 ele para, portanto, não executará mais o laço quando atingir este valor.

tabuada= int(input("Digite um numero para exibir a tabuada: "))

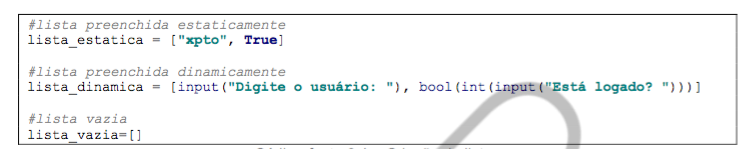
print("Tabuada do numero ", tabuada)

for valor in range (1,11,1):

print(str(tabuada)+ "x" + str(valor) + "=" + str((tabuada\*valor)))

----------------------------Capitulo 03- Listas e Funcoes\_ ----------------------------

Todas as listas podem ser inicializadas vazias ou com dados; caso você opte por criar uma lista vazia, deverá igualá-la, conforme é demonstrado no código abaixo.



Na lista\_dinamica - Como o *input* retorna uma *string*, devemos converter o dado para *int* (inteiro), para, então, posteriormente, convertê-lo para *bool* (booleano). Não podemos fazer a conversão diretamente de *string* para *bool*. O valor retornado será “False” somente se o número informado for igual a zero, qualquer outro valor trará o retorno “True”.

|  |  |
| --- | --- |
| inventario=[]  resposta="S"  while resposta=="S":  inventario.append(input("Equipamento: "))  inventario.append(input("Valor: "))  inventario.append(input("Numero Serial: "))  inventario.append(input("Departamento: "))  resposta=input("Digite \"S\" para continuar: ").upper()  for elemento in inventario:  print(elemento) | Geladeira  600  8574393483240234  Cozinha  Televisao  897  837438123923  Home |

Perceba no código acima que utilizamos o método append() para adicionar novos itens em nossa lista. Podemos afirmar que a cada passagem dentro do “while”, quatro novos dados serão adicionados na lista (nome do equipamento, valor do equipamento, número serial do equipamento e o nome do departamento onde se encontra o equipamento).

|  |  |
| --- | --- |
| equipamentos=[]  valores=[]  seriais=[]  departamentos=[]  resposta="S"  while resposta=="S":  equipamentos.append(input("Equipamento: "))  valores.append(input("Valor: "))  seriais.append(input("Numero Serial: "))  departamentos.append(input("Departamento: "))  resposta=input("Digite \"S\" para continuar: ").upper()  for equipamentos in equipamentos:  print("Equipamentos: " , equipamentos) | Equipamento: Geladeira Valor: 700 Numero Serial: 567899654 Departamento: home Digite "S" para continuar: s Equipamento: Televisao Valor: 655 Numero Serial: 543245433 Departamento: home Digite "S" para continuar: n  Equipamentos: Geladeira Equipamentos: Televisao |
| equipamentos=[]  valores=[]  seriais=[]  departamentos=[]  resposta="S"  while resposta=="S":  equipamentos.append(input("Equipamento: "))  valores.append(input("Valor: "))  seriais.append(input("Numero Serial: "))  departamentos.append(input("Departamento: "))  resposta=input("Digite \"S\" para continuar: ").upper()  for indice in range(0,len(equipamentos)):  print("\nEquipamento..: ", (indice+1))  print("Nome...........: ", equipamentos[indice])  print("Valor...........: ", valores[indice])  print("Serial..........: ",seriais[indice])  print("Departamento....: ", departamentos[indice]) | Equipamento: Roteador Valor: 4500 Numero Serial: 12345 Departamento: RH Digite "S" para continuar: S Equipamento: Impressora  Valor: 880 Numero Serial: 122345 Departamento: RH Digite "S" para continuar: n  Equipamento..: 1 Nome...........: Roteador Valor...........: 4500 Serial..........: 12345 Departamento....: RH  Equipamento..: 2 Nome...........: Impressora  Valor...........: 880 Serial..........: 122345 Departamento....: RH |

A estrutura do nosso “for” mudou, agora não estamos trabalhando com base nos elementos diretamente, mas, sim, de acordo com o índice. Para a variável “índice” que criamos no “for”, será atribuído o valor de 0 até a quantidade de elementos que existirem dentro da nossa lista “equipamentos” (função “len()”), que obviamente será a mesma quantidade de elementos que existirão nas listas: valores, seriais e departamentos,

|  |  |
| --- | --- |
| equipamentos=[]  valores=[]  seriais=[]  departamentos=[]  resposta="S"  while resposta=="S":  equipamentos.append(input("Equipamento: "))  valores.append(input("Valor: "))  seriais.append(input("Numero Serial: "))  departamentos.append(input("Departamento: "))  resposta=input("Digite \"S\" para continuar: ").upper()  for indice in range(0,len(equipamentos)):  print("\nEquipamento..: ", (indice+1))  print("Nome...........: ", equipamentos[indice])  print("Valor...........: ", valores[indice])  print("Serial..........: ",seriais[indice])  print("Departamento....: ", departamentos[indice])  busca=input("\nDigite o nome do equipamento que deseja buscar: ")  for indice in range(0,len(equipamentos)):  if busca==equipamentos[indice]:  print("Valor...: ", valores[indice])  print("Serial..: "), seriais[indice]) | Equipamento: Impressora  Valor: 600  Numero Serial: 567890  Departamento: RH  Digite "S" para continuar: S  Equipamento: Computador  Valor: 445  Numero Serial: 234321  Departamento: ADM  Digite "S" para continuar: n  Equipamento..: 1  Nome...........: Impressora  Valor...........: 600  Serial..........: 567890  Departamento....: RH  Equipamento..: 2  Nome...........: Computador  Valor...........: 445  Serial..........: 234321  Departamento....: ADM  Digite o nome do equipamento que deseja buscar: Computador  Valor...: 445  Serial..: 234321 |

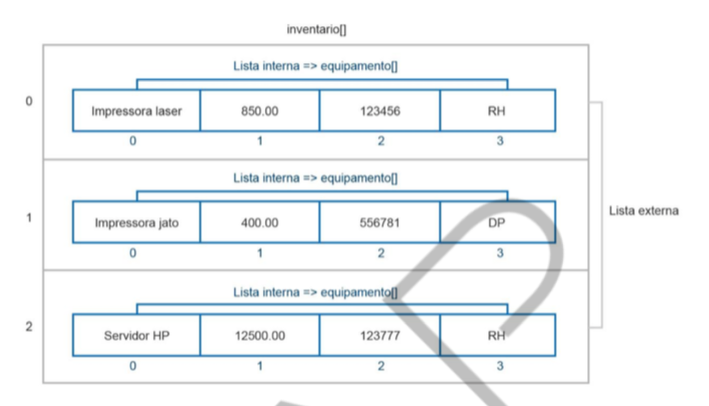
Vamos imaginar algumas outras situações:

 **Situação 1:** todos os equipamentos “impressora” receberão uma depreciação (desvalorização após certo período) de 10%. Monte o código que seria responsável por alterar o valor de todos os equipamentos “impressora”.

 **Situação 2:** um equipamento com um determinado número serial foi danificado e será descartado. Precisamos eliminar esse equipamento. **Dica:** para eliminar um item de uma lista, você utilizará o comando “del”. Exemplo: del lista[<indice>]

|  |
| --- |
| equipamentos=[]  valores=[]  seriais=[]  departamentos=[]  resposta="S"  while resposta=="S":  equipamentos.append(input("Equipamento: "))  valores.append(input("Valor: "))  seriais.append(input("Numero Serial: "))  departamentos.append(input("Departamento: "))  resposta=input("Digite \"S\" para continuar: ").upper()  for indice in range(0,len(equipamentos)):  print("\nEquipamento..: ", (indice+1))  print("Nome...........: ", equipamentos[indice])  print("Valor...........: ", valores[indice])  print("Serial..........: ",seriais[indice])  print("Departamento....: ", departamentos[indice])  busca=input("\nDigite o nome do equipamento que deseja buscar: ")  for indice in range(0,len(equipamentos)):  if busca==equipamentos[indice]:  print("Valor...: ", valores[indice])  print("Serial..: ", seriais[indice])  depreciacao=input("\Digite o equipamento que sera depreciado: ")  for indice in range(0,len(equipamentos)):  if depreciacao==equipamentos[indice]:  print("Valor antigo: ",valores[indice])  valores[indice] = valores[indice]  print("Novo valor: ",valores[indice])  serial=int(input("\nDigite o serial do equipamento que sera excluido: "))  for indice in range(0,len(departamentos)):  if seriais[indice]==serial:  del departamentos[indice]  del equipamentos[indice]  del seriais[indice]  del valores[indice]  break  for indice in range(0,len(equipamentos)):  print("\nEquipamento..: ", (indice+1))  print("Nome...........: ", equipamentos[indice])  print("Valor...........: ", valores[indice])  print("Serial..........: ",seriais[indice])  print("Departamento....: ", departamentos[indice]) |

Listas dentro de listas



De acordo com a Figura 3.4, iremos definir que a lista “inventario[]” é a lista externa e armazenará nas suas posições uma lista chamada “equipamento[]”, que, por sua vez, fará o papel de lista interna. De acordo com a imagem, se utilizarmos os prints abaixo:

*   print (inventario[0][2]) => será exibido o dado “123456”, porque ele retornará o elemento 0 da lista exterior e, dentro desse elemento, 0 irá buscar o dado que estiver na posição 2 da lista interior, ou seja, o serial (elemento 2) da impressora a laser. Outro exemplo:
*   print (inventario[1][3]) => irá retornar “DP”, que é o conteúdo da posição 3 do segundo elemento da lista “inventario”.
* Caso você não tenha uma tendência em preferir códigos mais complexos, irá perceber que o código do arquivo “ListasDentroDeListas.py” se tornou mais legível que o anterior. A única regra é saber que, na posição 0, teremos o nome do equipamento; na posição 1, o valor; na posição 2, o serial; e na posição 3, o departamento.
* inventario=[]
* resposta="S"
* while resposta=="S":
* equipamento=[input("Equipamento: "),
* float(input("Valor: ")),
* int(input("Numero Serial: ")),
* input("Departamento: ")]
* inventario.append(equipamento)
* resposta=input("Digite \"S\" para continuar: ").upper()
* for elemento in inventario:
* print("Nome........: ",elemento[0])
* print("Valor...........: ",elemento[1])
* print("Serial..........: ",elemento[2])
* print("Departamento....: ",elemento[3])
* busca=input("\nDigite o nome do equipamento que deseja buscar: ")
* for elemento in inventario:
* if busca==elemento[0]:
* print("Valor...: ", elemento[1])
* print("Serial..: ", elemento[2])
* depreciacao=input("\Digite o equipamento que sera depreciado: ")
* for elemento in inventario:
* if depreciacao==elemento[0]:
* print("Valor Antigo...: ", elemento[1])
* elemento[1] = elemento[1]\*0.9
* print("Novo valor..: ", elemento[1])
* serial=int(input("\nDigite o serial do equipamento que sera excluido: "))
* for elemento in inventario:
* if elemento[2]==serial:
* inventario.remove(elemento)
* for elemento in inventario:
* print("Nome........: ",elemento[0])
* print("Valor...........: ",elemento[1])
* print("Serial..........: ",elemento[2])
* print("Departamento....: ",elemento[3])

Funções para listas numéricas

* inventario=[]
* resposta="S"
* while resposta=="S":
* equipamento=[input("Equipamento: "),
* float(input("Valor: ")),
* int(input("Numero Serial: ")),
* input("Departamento: ")]
* inventario.append(equipamento)
* resposta=input("Digite \"S\" para continuar: ").upper()
* for elemento in inventario:
* print("Nome........: ",elemento[0])
* print("Valor...........: ",elemento[1])
* print("Serial..........: ",elemento[2])
* print("Departamento....: ",elemento[3])
* busca=input("\nDigite o nome do equipamento que deseja buscar: ")
* for elemento in inventario:
* if busca==elemento[0]:
* print("Valor...: ", elemento[1])
* print("Serial..: ", elemento[2])
* depreciacao=input("\Digite o equipamento que sera depreciado: ")
* for elemento in inventario:
* if depreciacao==elemento[0]:
* print("Valor Antigo...: ", elemento[1])
* elemento[1] = elemento[1]\*0.9
* print("Novo valor..: ", elemento[1])
* serial=int(input("\nDigite o serial do equipamento que sera excluido: "))
* for elemento in inventario:
* if elemento[2]==serial:
* inventario.remove(elemento)
* for elemento in inventario:
* print("Nome........: ",elemento[0])
* print("Valor...........: ",elemento[1])
* print("Serial..........: ",elemento[2])
* print("Departamento....: ",elemento[3])
* valores=[]
* for elemento in inventario:
* valores.append(elemento[1])
* if len(valores)>0:
* print("O equipamento mais caro custa: ",max(valores))
* print("O equipamento mais barato custa: ", min(valores))
* print( "O total de equipamentos é de: ", sum(valores))

Funções para listas numéricas

Para criarmos funções, utilizaremos o comando “def”, e a estrutura básica de uma função é a seguinte:

def <identificador da funcao> (<parametro(s)>): <código que será executado> return <Dado que será retornado, caso seja necessário>

Onde:

*   O identificador da função deve seguir as mesmas regras e padronizações dos identificadores das variáveis e listas. As funções representam “ações” como: inserir, exibir, consultar, apagar, calcular, então, é de bom-tom utilizar um verbo no identificador da função.

  O parâmetro é um dado que será fornecido para que a função possa executar o seu bloco de códigos. É como se fossem os ingredientes de uma receita, por exemplo, para que uma função possa calcular uma média aritmética entre duas notas, você deverá fornecer as duas notas ou ainda   para que uma função calcule o salário líquido de um colaborador, precisará informar para a função, no mínimo, o salário bruto.

*   O código a ser executado representa o conjunto de códigos que possuem uma mesma finalidade dentro da aplicação.
*   A última linha de “return” é opcional e deve ser usada somente quando você desejar que a função retorne um valor para o módulo principal.

Nome do arquivo -🡪 IdentificacaoDeFuncoes.py

def preencherInventario(lista):

resp="S"

while resp=="S":

equipamento=[input("Equipamento: "),

float(input("Valor: ")),

int(input("Numero Serial: ")),

input("Departamento: ")]

lista.append(equipamento)

resp=input("Digite \"s\" para continuar: ").upper()

def exibirInventario(lista):

for elemento in lista:

print("Nome......", elemento[0])

print("Valor......", elemento[1])

print("Serial......", elemento[2])

print("Departamento......", elemento[3])

def localizarPorNome(lista):

busca=input("Digite o nome do equipamento que deseja buscar: ")

for elemento in lista:

if busca==elemento[0]:

print("Valor antigo: ", elemento[1])

elemento[1]=elemento[1]\* 0.9

print("Novo valor: ", elemento[1])

def depreciarPorNome(lista):

depreciar=input("Digite o nome do equipamento depreciado: ")

for elemento in lista:

if depreciar==elemento[0]:

print("Valor antigo: ",elemento[1])

elemento[1]=elemento[1]\*0.7

print("O novo valor eh: ", elemento[1])

def excluirPorSerial(lista):

serial=int(input("Digite o serial do equipamento que será excluido: "))

for elemento in lista:

if elemento[2]==serial:

lista.remove(elemento)

return "Itens excluidos."

def resumirValores(lista):

valores=[]

for elemento in lista:

valores.append(elemento[1])

if len(valores)>0:

print("O equipamento mais caro custa: ", max(valores))

print( "O equipamento mais barato custa ", min(valores))

print( "O total de equipamentos e de : ", sum(valores))

nome do arquivo🡪 ModuloPrincipal.py

from IdentificacaoDeFuncoes import\*

minhaLista=[]

print("Preenchendo")

preencherInventario(minhaLista)

print("Exibindo")

exibirInventario(minhaLista)

print("Pequisando")

localizarPorNome(minhaLista)

print("Alterando")

depreciarPorNome(minhaLista)

print("Excluindo")

print(excluirPorSerial(minhaLista))

exibirInventario(minhaLista)

print("Resumindo")

resumirValores(minhaLista)

------------------Capitulo 4 - Manipulação de Dicionários de dados e Tuplas----------------



Vamos analisar o código. Na primeira linha, estamos criando o dicionário de dados. Repare que, em vez de utilizarmos os colchetes (como fizemos com as listas), usamos as chaves “{}”, essa é a representação de um dicionário de dados.

Na segunda linha, preenchemos fielmente o nosso dicionário de acordo com o nosso Exemplo de um dicionário de dados. Acima, perceba que adotamos os logins como chave de cada objeto (no caso, são dois). Depois da chave, utilizamos dois pontos (:) e, então, surgem os dados da chave que foi preenchida. Como no exemplo, o login “Chaves” possui mais de um dado, optamos em criar uma lista para armazenar estes dados e, dentro desta lista, colocamos o nome, a data de último acesso e o nome da última estação onde esteve logado. Após a vírgula (,) iniciamos o outro objeto, com a chave “Quico”, seguido de dois pontos (:) e, então, a lista com os dados deste objeto. E aí encerramos o dicionário, fechando as chaves. Se, por um engano, você colocar dois objetos com a mesma chave, o segundo objeto irá sobrescrever o primeiro.

Também podemos adicionar itens no dicionário da seguinte forma:

**usuario ["Florinda"**]=[**"Florinda Flores"**, **"26/11/2017"**, **"Recep\_01"**]

A linha acima apresentada normalmente é utilizada quando queremos adicionar os objetos de maneira singular, ou seja, objeto por objeto

usuarios={} {} Dicionario

usuarios={

"Chaves": ["Chaves Silva", "17/06/2017", "Recep\_01"],

"Quico": ["Enrico Flores", "03/06/2017", "Raiox\_02"],

}

print(usuarios)

print("################=========################")

print("Dados: ",usuarios.get("Chaves"))

No primeiro print(), exibimos tudo o que existe dentro do dicionário. Já no último print(), somente os dados do objeto que tiver a chave “Chaves”. Isso foi possível porque invocamos o nosso dicionário “usuarios” e utilizamos o seu método get(), que recebe um dado e vai pesquisá-lo entre as chaves que existem dentro do dicionário, caso ele encontre, retornará os dados relativos à chave encontrada.

Vamos começar criando um arquivo, chamado: “ManagerUsers.py” e iniciar perguntando ao usuário o que ele deseja realizar: inserir, pesquisar, excluir, listar ou sair. Posteriormente vamos descartar tudo e utilizer funcoes:

usuarios={}

opcao=input("O que deseja realizar? \n"+

"<I> - Para Inserir um usuario \n"+

"<P> - Para Pesquisar um usuario \n"+

"<E> - Para Excluir um usuario \n"+

"<L> - Para Listar um usuario: ").upper()

while opcao == "I" or opcao=="P" or opcao =="E" or opcao=="L":

if opcao=="I":

chave=input("Digite o login: ").upper()

nome=input("Digite o nome: ").upper()

data=input("Digite a ultima data de acesso: ").upper()

estacao=input("Digite a ultima estacao acessada: ").upper()

usuarios[chave]=[nome,data,estacao]

# Poderia ser enxuto dessa forma:

# chave=input("Digite o login: ").upper()

# usuarios[chave]=[input("Digite o nome: ").upper(),

# input("Digite a ultima data de acesso: "),

# input("Digite a ultima estacao acessada: ").upper()

# Ou mais resumido ainda ( tirando a variavel "chave"):

# usuarios[input("Digite o login: ").upper()=[input("Digite o nome: ").upper(),

# input("Digite a ultima data de acesso: "),

# input("Digite a ultima estacao acessada: ").upper()]

opcao=input( "O que deseja realizar? \n" +

"<I> - Para Inserir um usuario \n"+

"<P> - Para Pesquisar um usuario \n"+

"<E> - Para Excluir um usuario \n"+

"<L> - Para Listar um usuario: ").upper()

Criando um arquivo Funcoes.py :

* def perguntar():
* resposta= input("O que deseja realizar? \n"+
* "<I> - Para Inserir um usuario \n"+
* "<P> - Para Pesquisar um usuario \n"+
* "<E> - Para Excluir um usuario \n"+
* "<L> - Para Listar um usuario: ").upper()
* return resposta
* def inserir(dicionario):
* dicionario[input("Digite o login: ").upper()]=[input("Digite o nome: ").upper(),
* input("Digite a ultima data de acesso: "),
* input("Digite a ultima estacao acessada: ").upper()]
* def pesquisar(dicionario, chave):
* lista=dicionario.get(chave)
* if lista!= None:
* print("Nome.........: " +lista[0])
* print("Ultimo acesso....: " + lista[1])
* print("Ultima estacao...: " +lista[2])
* def excluir(dicionario,chave):
* if dicionario.get(chave)!=None:
* del dicionario[chave]
* print("Objeto Eliminado")
* def listar(dicionario):
* for chave,valor in dicionario.items():
* print("Objeto......")
* print("Login: ", chave)
* print("Dados: ",valor)
* Na função pesquisar(), precisamos receber o dicionário (onde se pretende pesquisar) e a chave (o dado que será pesquisado). Logo após, vamos preencher uma lista com o resultado da pesquisa proveniente do uso da função get(). Verificamos se a lista não está vazia (!= - representa diferente), caso esta condição seja verdadeira, vamos exibir os três dados que compõem a lista. Teremos, na primeira posição (zero), o nome do usuário; na segunda posição (um), a última data de acesso; e na terceira posição (dois), a última estação acessada. Caso não encontre a chave, não será retornada nenhuma mensagem;
*   Na função excluir(), também recebemos o dicionário de onde o objeto será excluído e a chave do objeto que se deseja excluir. Efetivamente, antes da exclusão, devemos verificar se a chave existe. Por isso, verificamos com a função get() se será retornado algo diferente de vazio e, se isso for verdade, invocamos o comando “del”, que eliminará o objeto de acordo com a chave que foi recebida.
* Na função listar(), precisaremos apenas do dicionário que contém os dados que desejamos exibir, com isso, montamos um foreach, mas perceba que, desta vez, utilizamos dois valores (chave e valor) para que possamos dar uma saída um pouco mais “clean” para o nosso usuário final. Poderíamos fazer de outras formas, inclusive, utilizando a nossa outra função pesquisar().

Descartando o conteúdo do ManagerUsers.py” e utilizando Funcoes :

from Funcoes import\*

usuarios={}

opcao=perguntar()

while opcao=="I" or opcao=="P" or opcao=="E" or opcao=="L":

if opcao=="I":

inserir(usuarios)

opcao=perguntar()

if opcao =="P":

pesquisar(usuarios,input("Qual login deseja pesquisar?"))

if opcao=="E":

excluir(usuarios,input("Qual o login deseja excluir? "))

if opcao =="L":

listar(usuarios)

opcao=perguntar()

Veja como o código ficou muito mais limpo.

**Métodos adicionais para dicionários**

Os dicionários possuem alguns métodos que podem ser úteis em outras situações do dia a dia, vamos explorar alguns deles:

* **items()**: responsável por retornar em forma de lista os elementos do dicionário, para isso, ele fará uso do recurso de tuplas, que será apresentado no tópico seguinte. Basicamente, cada dupla terá uma chave e um dado, conforme podemos perceber no exemplo abaixo. Considerando nosso exemplo, poderíamos ter este método retornando os dados da seguinte forma:  [(“ologin”,[“nome”,”data”,”estacao”]),(“login2”,[“nome2”, “data2”,”est2”])]  Perceba que temos a lista e, dentro da lista, dois elementos entre parênteses (cada elemento é uma tupla) e, dentro de cada elemento, dois elementos, a chave (login) e o dado (lista com nome, data e estação). Este método é muito utilizado em foreachs, como podemos perceber no método listar() que criamos anteriormente.
* **values()**: podemos retornar também somente os dados, descartando as chaves, ou seja, esse método retornar uma lista formada apenas pelos dados. Se pegarmos como exemplo o nosso caso do histórico dos usuários, teremos uma lista dentro de outra lista, e o exemplo ficaria como o que está apresentado abaixo:  [ [“nome”,”data”,”estacao”],[“nome2”, “data2”,”est2”]]
* **keys()**: claro que se retornarmos somente os dados, podemos retornar também somente as chaves. E é isto o que este método faz, retornando todas as chaves do dicionário em forma de lista, conforme apresentado na linha abaixo:  [“ologin”, “login2”]
* **has\_key()**: este método permitirá que você tenha a resposta se a chave existe ou não dentro do dicionário, ele irá retornar True (1) ou False (0).
* **clear()**: esvazia completamente o dicionário.
* **popitem()**: este é um método próprio para quem deseja montar algum dicionário que contenha elementos que serão executados, de maneira aleatória, individualmente e, na sequência, deverão ser eliminados do dicionário. Poderíamos pensar em um dicionário com dicas, e à medida cada dica fosse exibida, automaticamente ela deveria ser retirada do dicionario.

Tuplas

As tuplas, como já dissemos, são estruturas também para dados voláteis, assim como as variáveis, listas e dicionários. Porém, você percebeu que cada uma possui as suas devidas aplicações, e com as tuplas não poderia deixar de ser diferente.

Uma característica única das tuplas é o fato de elas não aceitarem alteração sobre os dados que já estiverem nelas inseridos. Além disso, as tuplas sempre são representadas com seus dados entre parênteses. Então, podemos sintetizar que: **listas envolvem os dados entre colchetes; dicionários entre chaves; e tuplas entre parênteses.**

Normalmente, aplicamos mais o conceito de tuplas para realizar a leitura de uma resposta do Python, e não de alguém que inseriu dados. É o exemplo do método “items()” que vimos dos dicionários, ele retorna os dados do dicionário, identificando cada elemento como uma tupla, naturalmente aplicado para exibição, e foi retornado pelo Python, uma vez que o dado foi preenchido através do dicionário. No nosso exemplo anterior, utilizamos os dados do usuário na posição de dados do dicionário, e como chave utilizamos uma string, no caso, o nome do login. Se, por exemplo, quiséssemos atribuir como chave o e-mail de uma pessoa, e este e-mail estivesse dividido entre dois dados (usuário e servidor), não poderíamos colocá-lo na chave de um dicionário como lista, mas, sim, como tupla.

Outro exemplo: você poderia pensar em definir um endereço IP como chave de um dicionário, mas seria importante manter os dados divididos em octetos, a fim de identificar grupos que pertencem a uma mesma rede (normalmente os dois primeiros octetos) ou usuários que logaram em uma mesma máquina (normalmente os dois últimos octetos identificam a estação).

Para isso, você será obrigado a utilizar a estrutura de tuplas, que poderá armazenar os octetos separadamente e também ser implementada pelo dicionário na posição de chave.

ips={}

resp="S"

while resp=="S":

ips[(input("Digite os dois primeiros octetos: "),input("Digite os dois últimos octetos: "))]=input("Nome da máquina: ")

resp=input("Digite <S> para continuar: ").upper()

print("Exibindo ip ́s: ")

for ip in ips.keys():

print(ip[0]+"."+ip[1])

print("Exibindo máquinas com o mesmo endereço: ")

pesquisa=input("Digite os dois últimos octetos: ")

for ip,nome in ips.items():

print("Máquinas no mesmo endereço (redes diferentes)")

if(ip[1]==pesquisa):

print(nome)

print("Exibindo as máquinas que compõem uma mesma rede: ")

rede=input("Digite os dois primeiros octetos: ")

for ip,nome in ips.items():

if(ip[0]==rede):

print(nome)

Como podemos observar no código, criamos um dicionário chamado “ips”, uma variável chamada “resp” para controlar o nosso laço e, dentro do laço de repetição, vamos preencher o nosso dicionário, mas perceba que, na chave do dicionário, inserimos dois valores. A primeira parte do ip e a segunda parte do ip, dentro de uma tupla, e o dado do elemento que está sendo adicionado no dicionário é o nome da estação.

Vejamos agora como aproveitar os dados da chave no formato de tuple com o codigo abaixo:



A saída dos dados, após a execução do código, apresentou os ips concatenados com um ponto “.”, conforme a função print() que está dentro do laço “for”. Veja como unimos as duas partes da chave do dicionário na mesma linha da função print(), utilizando a posição dos dados, ou seja, recuperando a posição zero (ip[0]) e a posição um (ip[1]). Veja também que o nosso laço for está recuperando apenas as chaves do nosso dicionário, que é onde se encontra a nossa tupla.

Uma outra situação em que podemos utilizar as tuplas é quando usarmos a função “enumerate()” em uma lista. Mas quando precisaremos utilizar a função “enumerate()” em uma lista? Quando quisermos numerar cada componente da lista a fim de garantir que cada elemento dela poderá ser utilizado como dado chave de um dicionário.

usuarios={} # criamos Dicionario para usuario

resp="S"

emails=[] # listas

while resp=="S":

emails.append(input("Digite um e-mail: ").lower())

resp=input("Digite <S> para continuar: ").upper()

tupla=list(enumerate(emails))

for chave in range (0,len(tupla)):

print("Email: ", tupla[chave][1])

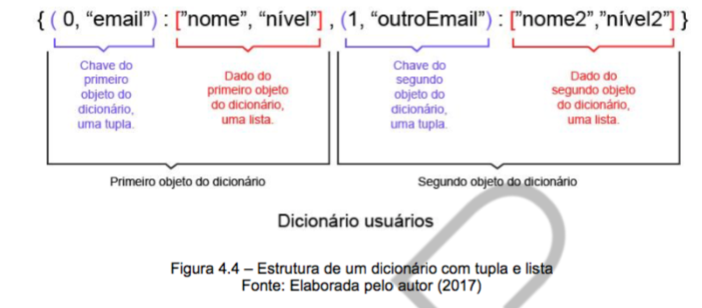
usuarios[tupla[chave]]=[input("Digite o nome"), input("Digite o nível")]

Na segunda linha do código acima, estamos enumerando (função enumerate()) cada item encontrado na lista “e-mails” e gerando uma tupla com cada elemento (função: list()), formados pelo número e pelo e-mail. Em seguida, utilizamos um laço “for” atrelado ao tamanho da nossa tupla, ou seja, à quantidade de e-mails que foram armazenados.

Se foram armazenados três e-mails, teremos que solicitar três nomes e três níveis e assim sucessivamente. Por isso, utilizamos a função range(), que controlará o laço “for” de zero até a quantidade de elementos encontrados na “tupla”. Dentro do laço “for”, vamos, primeiramente, exibir o e-mail que receberá o nome e o nível: tupla[chave][1] => da tupla e recuperar o elemento de acordo com o valor da variável-chave, ou seja, da primeira vez, ela estará valendo zero, por isso pegará o primeiro elemento da tupla.

Cada elemento da tupla é formado por um número (que está na posição zero dentro do elemento da tupla) e o e-mail (que está na posição um dentro do elemento da tupla). O que queremos exibir deste elemento é o e-mail que está na posição um, por isso, o número um entre colchete

Após a exibição do e-mail, vamos preencher a chave do dicionário “usuários” com o elemento da tupla já enumerado que foi exibido anteriormente e, então, pediremos o nome e o nível para preencher o dado do objeto do dicionário, lembrando que o nome e o nível estão em formato de lista. Poderíamos exemplificar o nosso dicionário com dois objetos, da seguinte forma:



Procure compreender bem o parágrafo anterior, use a imagem como base e perceba que utilizamos todos os recursos vistos até o momento em um único exemplo: variável, dicionário, lista e tupla. Cada um sendo explorado com a sua melhor característica: a simplicidade e a economia de recursos da variável (para controlar o laço); a estrutura perfeita do dicionário; a segurança da tupla pelo fato de não permitir alteração da chave do dicionário; e a flexibilidade da lista no dado do dicionário.

for chave, dado in usuarios.items():

print("Usuario...: ",chave[0])

print("Email....: ", chave [1])

print("Nome.....: ", dado[0])

print("Nivel.....", dado [1])

Observe os dados que estamos exibindo, aproveitando suas posições. Primeiro, mostramos os dois dados da chave do dicionário (numeração e e-mail) e, depois, os dois dados do dicionário (nome e nível). Prático, não é mesmo?

Para concluir este capítulo, vale explicar que a String é uma lista com característica de tupla. Uma verdadeira simbiose! Isso quer dizer o seguinte: quando você armazena uma String como “Defesa”, internamente, foi criada uma lista na qual cada elemento terá um caractere.

------------------ Capitulo 5 - Manipulação de Arquivos E JSON ----------------

Manipulação de arquivos

Para a manipulação dos arquivos, utilizaremos uma função denominada “open()”, que permite várias ações :

**open(“<caminhoDoArquivo><nomeDoArquivo>”, “w”)** => indica que você está abrindo um arquivo para o modo de escrita (w => write), ou seja, permite que você escreva nele, caso o arquivo já exista, ele será sobrescrito. É como se você adquirisse um novo “caderno”, completamente em branco.

**open(“<caminhoDoArquivo><nomeDoArquivo>”, “r”)** => com a letra “r” (read) no segundo parâmetro da função, você abrirá o arquivo somente em modo de leitura, isso permite que outra pessoa, em outro computador, possa abrir esse arquivo para edição, mas você poderá apenas “consumir” os dados que estiverem dentro desse arquivo. Nesse caso, podemos fazer a analogia com um livro, ou seja, você não irá preenchê-lo, tão pouco alterá- lo, apenas irá fazer a leitura.

* **open(“<caminhoDoArquivo><nomeDoArquivo>”, “a”)** => dessa forma, você poderá ler e escrever no arquivo especificado, como se fosse um diário, no qual você irá acrescentando conteúdos de acordo com algum evento, periodicidade ou qualquer outro advento do dia a dia, mas sempre no final do diário, ou seja, a principal ideia é seguir concatenando os dados, por isso, a letra “a” referindo-se à “*append*”(anexar). Um exemplo bem prático para isso seria o controle de um arquivo de log, tudo o que for adicionado será acrescentado ao final dos dados que já existem.
* **open(“<caminhoDoArquivo><nomeDoArquivo>”, “x”)** => permite criar um novo arquivo em modo exclusivo (*e****X****clusive*), ou seja, uma vez que você criou/abriu o arquivo, ninguém mais poderá abri-lo.. Caso você tente abrir um arquivo que já existe, será retornada uma falha.
* **open(“<caminhoDoArquivo><nomeDoArquivo>”, “t”)** => o arquivo que for aberto com o parâmetro “t”(text) irá retornar para o Python o seu conteúdo como *string*, diferentemente do parâmetro “b”, que retornaria os dados em formato binário e exigiria uma conversão para *string*, caso fosse necessário.
* Para um primeiro teste, apresentamos a utilização da função open() junto ao comando “with”, esse comando permitirá representar, dentro do bloco identado, o arquivo “teste.txt” por meio do “alias” arquivo. Outra grande vantagem que obteremos todas as vezes que formos utilizar a função open() combinada ao comando with é que o controle do encerramento do arquivo em memória ficará por conta do comando with, isso faz com que você não precise utilizar o método close(), tampouco ocorrerá o fato do arquivo ficar aberto na memória sem qualquer necessidade. Por isso, recomendo que utilize esse combinado open() + with, você não vai se arrepender!
* Exemplos

with open("teste.txt", "w") as arquivo:

arquivo.write("Nunca foi tao facil criar um arquivo")

incluir este segundo

with open("teste.txt", "w") as arquivo:

arquivo.write("\nContinuacao do texto")

Esse código irá gerar um novo arquivo e escrever “Nunca foi tão fácil criar um arquivo”. O arquivo será encerrado, em seguida, aberto novamente e, então, a frase “Continuação do texto” será escrita nele, mas, ao abri-lo, você irá encontrar apenas a segunda frase, isso porque o primeiro arquivo foi sobrescrito e não concatenado. Essa é a função da opção “a”, que podemos utilizar no segundo parâmetro, altere somente no segundo *with* o “w” por “a” e veja como irá funcionar melhor agora. Vamos abrir um novo arquivo e gerar um *html* simples, para que você possa ter a noção exata do quão importante pode ser a geração de um arquivo dentro de uma linguagem de programação. Crie um novo arquivo chamado “ArquivoHTML.py” e monte o seguinte código:

with open("pagina.html", "w") as pagina:

pagina.write("<body><h1> Esta é uma pagina WEB </h1>")

pagina.write("<br><h2> Abaixo seguem alguns nomes importantes para o projeto: </h2>")

pagina.write("<h3>")

nome=" "

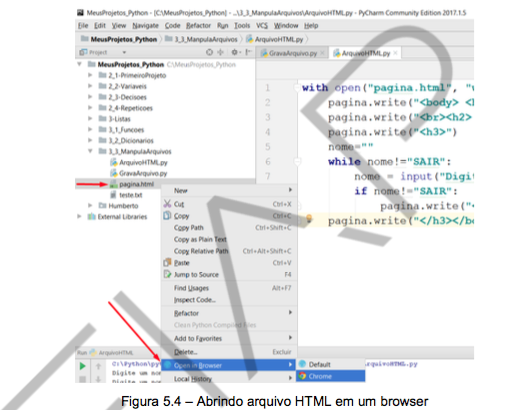
while nome!="SAIR":

nome=input("Digite um nome ou SAIR: ").upper()

if nome!="SAIR":

pagina.write("<br>"+nome)

pagina.write("</h3></body>")





Realizando a leitura do arquivo

Nesse código, abrimos o arquivo, teste.txt, para leitura (“r”) e atribuímos, para a variável “conteudo”, todo o conteúdo do arquivo (quando utilizamos o método **read()**), depois, imprimimos o tipo da variável e, na última linha, imprimimos o conteúdo do arquivo que está dentro da variável. O tipo de dado da variável “conteudo” que será exibido é “str”, ou seja, string, isso porque, quando não definimos o modo de saída, o valor-padrão é o “t”(text), por isso “r” ou “rt” ou “r+t” como segundo parâmetro retornarão o mesmo tipo. Podemos mudar para que a saída seja em bytes, para isso, substitua “r” por “rb” ou “r+b” e execute novamente esse bloco de código; verá que não só o tipo mudará para “byte” como também o conteúdo do arquivo a ser exibido terá uma formatação diferente, por *byte*. O mais comum mesmo é fazê-lo utilizando a saída no formato texto, não é à toa que ele é a forma-padrão.

Em “r”- str

with open("teste.txt", "r") as arquivo:

conteudo=arquivo.read()

print("Tipo de dado da variavel", type(conteudo))

print("\nConteudo do arquivo: \n", conteudo)

Tipo de dado da variavel <class 'str'>

Conteudo do arquivo:

Continuacao do texto

Em “rb - bytes

with open("teste.txt", "rb") as arquivo:

conteudo=arquivo.read()

print("Tipo de dado da variavel", type(conteudo))

print("\nConteudo do arquivo: \n", conteudo)

Tipo de dado da variavel <class 'bytes'>

Conteudo do arquivo:

b'\nContinuacao do texto'

with open("teste.txt", "rb") as arquivo:

conteudo=arquivo.readlines()

print("Tipo de dado da variavel", type(conteudo))

print("\nConteudo do arquivo: \n", conteudo)

Será muito útil a função **readlines()** quando optarmos por quebrar um grande arquivo de texto em partes, para que possamos retirar somente os dados que nos interessarem. Por falar nisso, vamos para um exemplo prático, no qual poderemos explorar mais ainda os conceitos, funções e métodos apresentados até o momento.

Gerando uma rotina para inventário

Vamos criar um arquivo “Inventario.py”e armazenar os seguintes dados: número patrimonial do ativo, a descrição do ativo, data da última atualização e o nome do departamento em que está localizado.

Primeiro iremos definir uma estrutura de dados para armazená-lo; eles serão recebidos pelo colaborador responsável por catalogar os ativos, e, então, persistiremos os dados para um arquivo, para que possam ser recuperados, “backupeados”, alterados, excluídos e estejam disponíveis para qualquer outra consulta que possa ser necessária posteriormente.

Utilizaremos uma estrutura de um dicionário de dados para armazená-los enquanto estiverem na condição de dados voláteis, em que a chave será o número patrimonial do ativo (que não pode ser repetido). Sobre essa chave, teremos uma lista com o departamento, data da última alteração e descrição. Após a gerência dos dados no dicionário, iremos persistir em um arquivo do tipo “csv”, que representa um padrão de arquivo no qual os dados poderão ser abertos dentro do Excel e de outros programas que aceitam esse tipo de arquivo. Isso irá garantir maior flexibilidade e portabilidade sobre os dados persistidos. Agora, vamos montar o seguinte código:

inventario={}

opcao =int(input("Digite: "

"\n<1> para registrar ativo"

"\n<2> para persistir em arquivo"

"\n<3> para exibir ativos armazenados: "))

while opcao>0 and opcao <4:

if opcao ==1:

resp="S"

while resp=="S":

inventario[input("Digite o numero patrimonial: ")]=[

input( "Digite a data da ultima atualizacao: "),

input("Digite a descricao: "),

input( "Digite o departamento: ")]

resp=input( "Digite <S> para continuar ").upper()

elif opcao==2:

with open("inventario.csv", "a") as inv:

for chave, valor in inventario.items():

inv.write(chave + ";" + valor[0]+ ";" + valor[1]+";"+valor[2]+"\n")

print ("Persistindo com sucesso!")

elif opcao==3:

with open( "inventario.csv", "r") as inv:

print(inv.readlines())

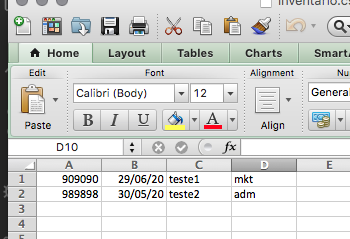
opcao= int(input("Digite: "

"\n<1> para registrar ativo"

"\n<2> para registrar em arquivo"

"\n<3> para exibir ativos armazenados: "))

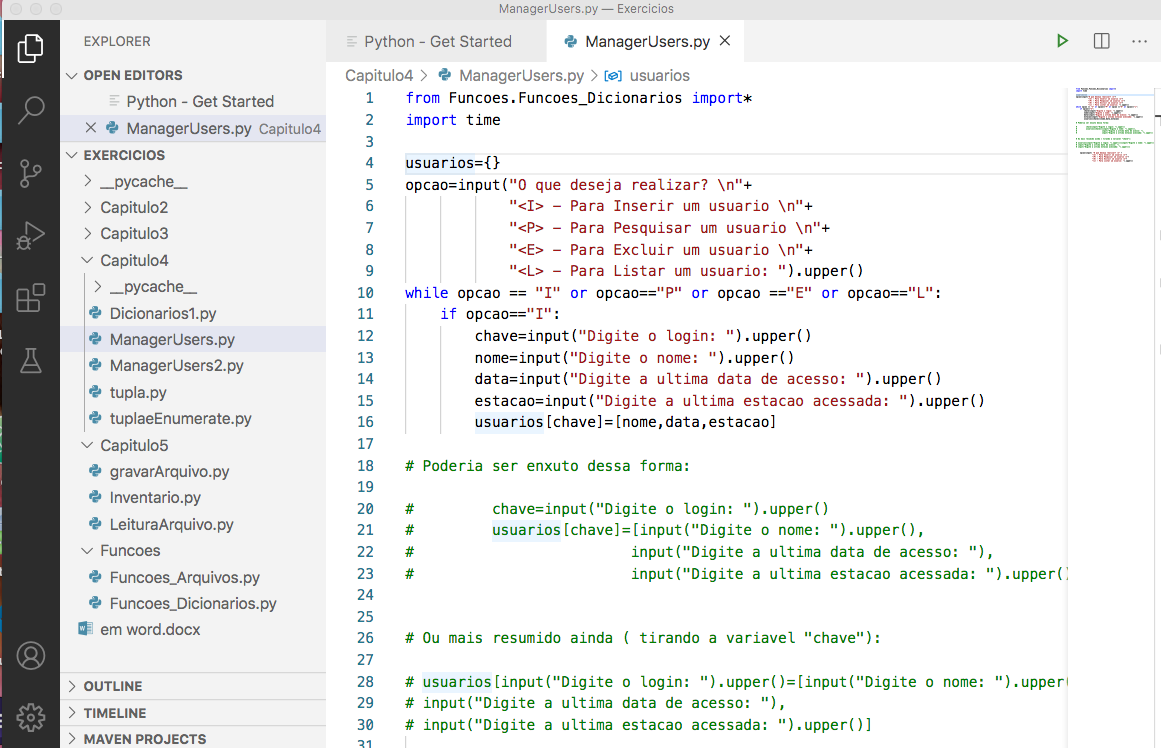
Foi criado um arquivo .csv



Seguiremos aprimorando nossa estrutura de projeto e, em vez de criarmos simplesmente um arquivo com as funções, vamos criar um “Python Package” chamado “Funcoes” e, dentro desse pacote, iremos criar todos os nossos arquivos que armazenarão funções.

No arquivo ManagersUser.py acrescentar 🡪

Pronto, dessa forma, todos os nossos arquivos que armazenarem funções estarão localizados em um mesmo pacote, o que nos facilitará reaproveitar funções entre projetos diferentes e também padronizar as importações.



Agora, dentro do arquivo “Funcoes\_Arquivos.py”, monte o código seguinte:

def chamarMenu():

escolha=int(input("Digite: "

"\n<1> para registrar ativo"

"\n<2> para persistir em arquivo"

"\n<3> para exibir ativos armazenados: "))

return escolha

def registrar(dicionario):

resp="S"

while resp=="S":

dicionario[input("Digite o numero patrimonial: ")]=[

input( "Digite a data da ultima atualizacao: "),

input("Digite a descricao: "),

input( "Digite o departamento: ")]

resp=input( "Digite <S> para continuar ").upper()

def persistir(dicionario):

with open("inventario.csv", "a") as inv:

for chave, valor in dicionario.items():

inv.write(chave + ";" + valor[0]+ ";" + valor[1]+";"+valor[2]+"\n")

return "Persistindo com sucesso!"

def exibir():

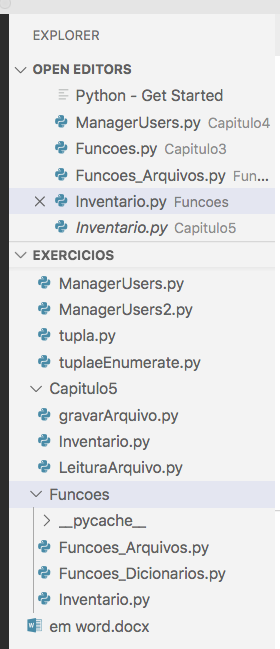
with open( "inventario.csv", "r") as inv:

linhas=(inv.readlines())

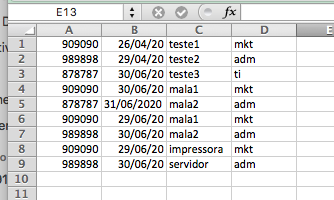
return linhas

Foram criadas, no código apresentado, quatro funções:

*   A função chamarMenu() apenas exibe para o usuário as opções que ele possui para a manipulação do inventário e retorna o valor selecionado por meio da variável “escolha”;
*   A função registrar() irá receber o dicionário responsável por armazenar os ativos e, então, irá preenchê-lo, enquanto o usuário digitar “S”;
*   A função persistir() ficou encarregada de descarregar o dicionário que for passado como parâmetro para dentro do arquivo “inventario.csv”; e
*   A função exibir() irá, finalmente, retornar o conteúdo das linhas do arquivo “inventario.csv” por meio da variável “linhas”.
* Agora, vamos atualizar o nosso arquivo “Inventario funcoes.py”, com o seguinte código:
* -🡪 From Funcoes.Funcoes\_Arquivos import\* ( se tiver em outra pasta.....)



* from Funcoes\_Arquivos import \*
* inventario={}
* opcao =chamarMenu()
* while opcao>0 and opcao <4:
* if opcao ==1:
* registrar(inventario)
* elif opcao==2:
* persistir(inventario)
* elif opcao==3:
* resultado=exibir()
* for linha in resultado:
* print(linha[2:-1])
* opcao=chamarMenu()
* O código do nosso arquivo “Inventario.py” tornou muito mais simples e muito mais fácil aplicar alterações e/ou manutenções. Uma atenção para a linha na qual chamamos o método exibir(), pois o chamamos dentro de um print() que será responsável por exibir tudo o que método exibir() retornar.
* Primeiro, atribuímos para a variável “resultado” os dados que forem retornados pela função exibir(), ou seja, o conteúdo do arquivo “inventario.csv” em forma de lista.
*   Montamos um “for” para percorrer toda a lista, isso porque não queremos que saia o número do patrimônio em todos os ativos.
*   De cada linha, iremos exibir somente do terceiro caractere [2] até o último [- 1]. Repare que fizemos esse recorte no elemento da lista por meio do recurso “[2:-1]”, que se chama “*slice*” e representa um recorte em uma String. Cada elemento da lista é uma string, e, por isso, podemos fatiá-la e exibir somente o que desejamos. Toda string tem, para o seu primeiro caractere, a posição 0, logo, estamos descartando a posição 0 (número do patrimônio) e a posição 1 (o “;”), exibindo do caractere 2 (primeiro dígito da data da última atualização) até o final, que é representado pelo -1. Logo, a primeira linha do meu arquivo está com:



No terminal :

<1> para registrar ativo  
<2> para persistir em arquivo  
<3> para exibir ativos armazenados: 1

Digite a data da ultima atualizacao: 29/06/2020  
Digite a descricao: impressora  
Digite o departamento: mkt  
Digite o numero patrimonial: 909090

Digite:   
<1> para registrar ativo  
<2> para persistir em arquivo  
<3> para exibir ativos armazenados: 2

Digite:

<1> para registrar ativo  
<2> para persistir em arquivo  
<3> para exibir ativos armazenados: 3

9090;26/04/2020;teste1;mkt  
9898;29/04/2020;teste2;adm  
8787;30/06/2020;teste3;ti  
9090;30/06/2020;mala1;mkt  
8787;31/06/2020;mala2;adm  
9090;29/06/2020;mala1;mkt  
9898;30/06/2020;mala2;adm  
9090;29/06/2020;impressora;mkt  
9898;30/06/2020;servidor ;adm

Manipulando arquivos de terceiros

Com o arquivo **"economic-indicators.csv"**

* Deveremos buscar as respostas às seguintes questões:
* - Qual o total de voos internacionais que partiram do aeroporto de Logan no ano de 2014?
* - Quando (mês/ano) ocorreu o maior trânsito de passageiros no aeroporto de Logan?
* - Qual o total de passageiros que passaram pelo aeroporto de Logan, no ano que for especificado pelo usuário?
* - Qual o mês que possui a maior média da diária de um hotel, com base no ano especificado pelo usuário?

Arquivo Boston.py

with open ("economic-indicator.csv", "r") as boston:

total=0

for linha in boston.readlines()[1:-1]:

total=total+float(linha.split(",")[3])

print("O total de voos é: ",total)

Detalhando um pouco o código apresentado:

*   Na primeira linha, abrimos o arquivo em modo somente de leitura;
*   Criamos uma variável para armazenar o total de voos;
*   Montamos um laço “foreach” para percorrermos somente da linha um “1” até a última linha válida, descartando a linha zero “0” (em que temos apenas os títulos);
*   Atribuímos, na variável total, o valor do terceiro elemento da lista que foi criada por meio do split(), separando os elementos por vírgula(“,”);
*   Imprimimos o conteúdo da variável, quando o laço for encerrado e a variável “total” tiver acumulado todos os valores do arquivo.
* Agora, complemente o arquivo resolvendo as outras questões, procure resolver uma a uma e siga abaixo para ver a solução proposta para a versão final do nosso arquivo “Boston.py”.

with open ("economic-indicators.csv", "r") as boston:

total\_voos=0

maior=0

total\_passageiros=0

maior\_media\_diaria=0

ano\_usuario=input("Qual o ano deseja pesquisar? ")

for linha in boston.readlines()[1:-1]:

lista=linha.split(",")

total\_voos=total\_voos+float(linha[3])

if float(lista[2])>float(maior):

maior=lista[2]

ano=lista[0]

mes=lista[1]

if ano\_usuario==lista[0]:

total\_passageiros=total\_passageiros+float(lista[2])

if float(lista[5])>float(maior\_media\_diaria):

maior\_media\_diaria=lista[5]

mes\_maior\_diaria=lista[1]

print("O total de voos é: ",total\_voos)

print("O mes/ano de maior movimento no aeroporto foi: ", str(mes), "/",str(ano))

print("O total de passageiros do ano", ano\_usuario, "foi de ", total\_passageiros)

print( "O mes do ano ",ano\_usuario, "com maior media para diaria de hotel foi de ", mes\_maior\_diaria)

No terminal

Qual o ano deseja pesquisar? 2013  
O total de voos é: 84.0  
O mes/ano de maior movimento no aeroporto foi: 7 / 2014  
O total de passageiros do ano 2013 foi de 30237699.0  
O mes do ano 2013 com maior media para diaria de hotel foi de 10

Um caminho para a portabilidade

Abordaremos outro padrão, também muito utilizado, para que você garanta que a saída dos dados da sua aplicação tenha portabilidade, esse é o tal JSON!

**JSON** (JavaScript Object Notation)

JSON é uma sigla, **J**ava**S**cript **O**bject **N**otation, que representa um padrão para armazenamento de dados, cuja principal função é permitir a troca de dados entre aplicações e plataformas distintas. O Python possui um módulo para facilitar o trabalho com esse tipo de padronização. Vamos a um exemplo prático, crie um arquivo chamado “Manipular\_JSON.py” e digite o código a seguir:

import json

inventario={}

opcao=int(input("Digite: "

"\n<1> para registrar ativo"

"\n<2> para exibir ativos armazenados: "))

while opcao>0 and opcao<3:

if opcao==1:

resp="S"

while resp=="S":

inventario[input("DIgite o numero patrimonial: ")]=[

input("Digite a data da ultima atualizacao: "),

input("Digite a descricao: "),

input("Digite o departamento: ")]

resp=input("Digite <S> para continuar. ").upper()

with open("inventario\_json.json", "w") as arq\_json:

json.dump(inventario, arq\_json)

print("Json gerado!!!")

elif opcao==2:

with open("inventario\_json.json", "r") as arq\_json:

resultado=json.load(arq\_json)

for chave, dado in resultado.items():

print("Data..........: ", dado[0])

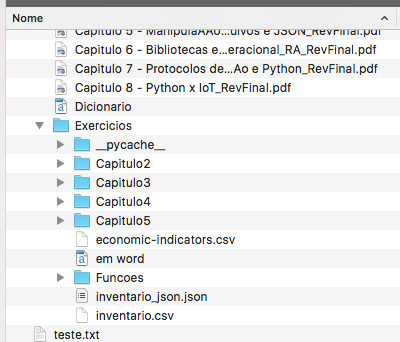
print("Descricao.....: ", dado[1])

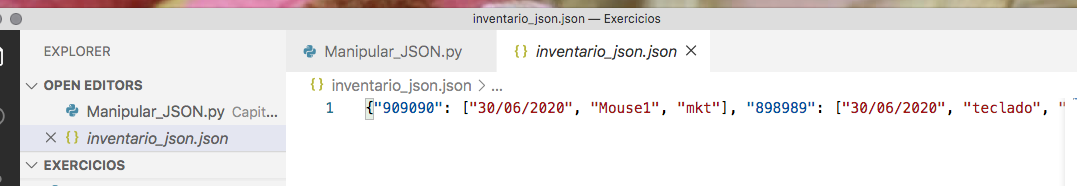
print("Departamento..: ", dado[2])

opcao = int(input("Digite: "

"\n<1> para registrar ativo"

"\n<2> para exibir ativos armazenados: "))





Como você já percebeu, o exemplo adotado foi o mesmo do tópico anterior, para que você possa ter um parâmetro entre o código que manipula um arquivo “txt”, “csv” ou “html” e o código que utiliza o padrão JSON.

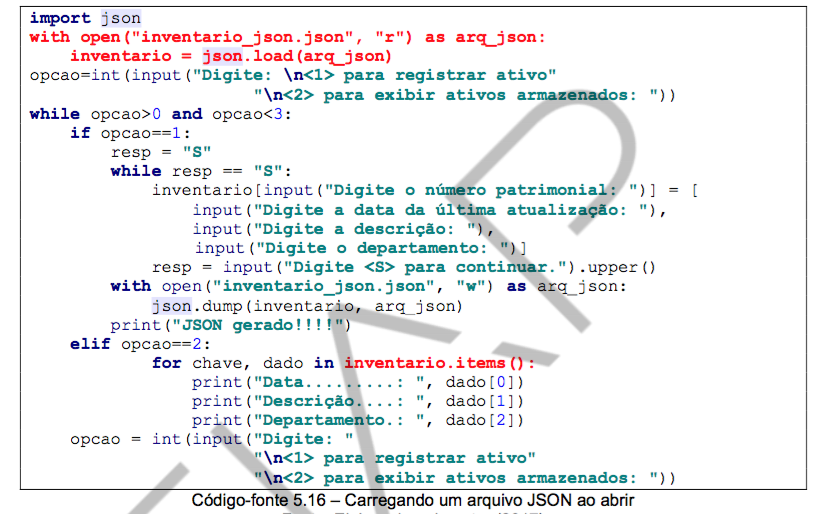
O primeiro detalhe a ser observado está na primeira linha do código, o “import”. Como já afirmamos anteriormente, o Python possui um módulo para a manipulação de JSON e, para que possamos utilizar os seus objetos e métodos, devemos importá- lo.

Perceba que, nesse exemplo, retiramos a antiga opção “2”, que era responsável por persistir no arquivo, pois agora realizamos a persistência logo após o usuário digitar todos os dados no dicionário “inventario”, assim que o loop for encerrado, já realizamos a persistência no formato JSON.

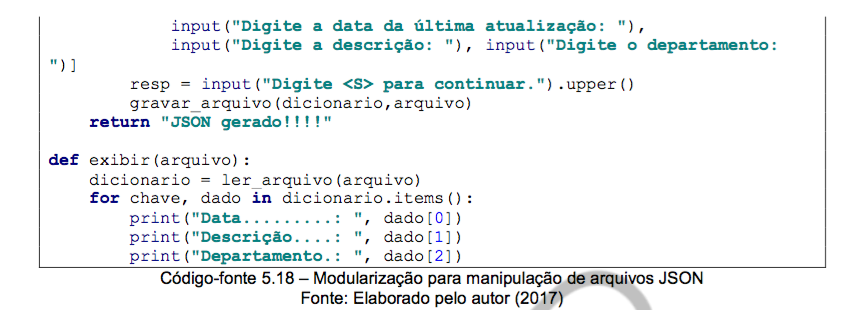
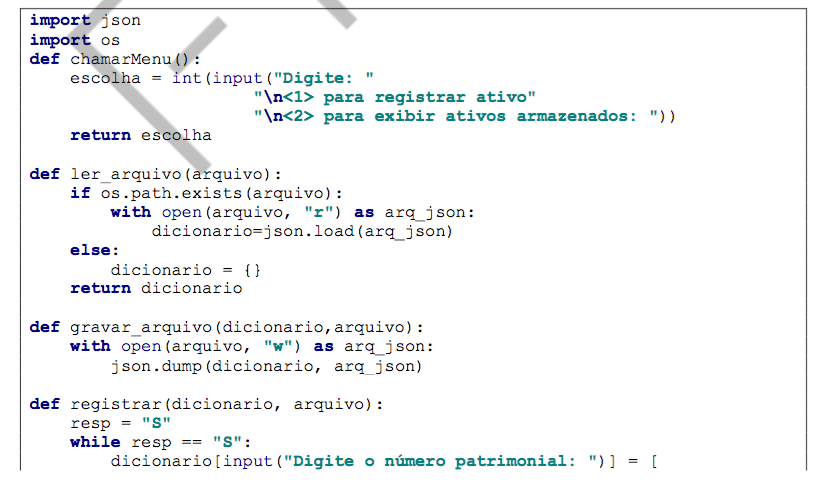
Para gravarmos o dicionário “inventario” no arquivo, utilizamos o método dump(), que pertence ao objeto “json”. Esse método é formado, basicamente, por dois parâmetros: o que será gravado, no nosso caso, o dicionário “inventario” e onde será gravado, quando definimos por meio do alias “arq\_json”, que aponta para o arquivo “inventario\_json.json”, o qual estará aberto em modo escrito, ou seja, caso ele não exista será criado automaticamente e, caso já exista, será sobrescrito.

Continuando a explicação do código, na opção “2”, abrimos o nosso arquivo em modo de leitura apenas e, então, chamamos o método load(), que também pertence ao objeto “json”. Ele descarrega o arquivo informado no parâmetro para uma variável, no nosso exemplo, a variável “resultado”. Em seguida, montamos um laço “foreach”, que irá trazer a chave (número patrimonial) e os dados (uma lista com data, descrição e departamento); finalizamos com a exibição e o menu de opções novamente.

Agora perceba que, como não utilizamos, na opção “1”, o modo de concatenação, se você executar novamente o seu código, verá que o seu arquivo anterior será sobrescrito, isto é, se você lançar um ativo, o seu arquivo ficará apenas com um ativo e não mais com três (considerando os outros dois que você preencheu anteriormente). Para solucionar isso, faremos uma alteração na estrutura do nosso código, ao invés de criarmos o dicionário “inventario” vazio, já o criaremos preenchido pelo arquivo, assim, a nossa opção “1” irá adicionar mais um objeto ao dicionário, que já estará preenchido com o conteúdo do arquivo e, finalmente, quando for realizar a gravação, irá gravar todos os objetos que estiverem no dicionário. Veja como ficará o código:



As alterações estão destacadas pela cor vermelha. Criamos o nosso dicionário “inventario”, já o preenchendo com o conteúdo do arquivo “inventario\_json.json”. Na opção “2”, de exibição, basta utilizarmos o dicionário “inventario” no “foreach”, assim percorreremos todos os dados de maneira atualizada.



Nesse arquivo, criamos algumas rotinas tão abstratas que poderiam funcionar com qualquer aplicação que tivesse a função de manipular arquivos JSON, é o caso das funções: ler\_arquivo() e gravar\_arquivo(). Para qualquer situação envolvendo arquivos JSON, elas podem ser reaproveitadas, tanto que as próprias funções registrar() e exibir() utilizam essas funções.

Agora sim, podemos pensar no nosso arquivo “Manipular\_JSON.py”, que ficará muito mais “*clean*”, conforme podemos perceber no código a seguir:

