Listas, Dicionários, Tuplas e Conjuntos

Lista

Listas são coleções heterogêneas de objetos, que podem ser de qualquer tipo, inclusive outras listas.

As listas no Python são **mutáveis**, podendo ser alteradas a qualquer momento. Listas podem ser fatiadas da mesma forma que as ***strings***, mas como as listas são mutáveis, é possível fazer atribuições a itens da lista.

```
lista = [a, b, ..., z]
```

```
#exemplo de listas
lista1=[1,2,3,4]
lista2=['python','java','c#']
lista3 = [1,2,'python', 3.5, 'java']
lista4 = list('python')
```

Vejamos um exemplo em que um aluno tem cinco notas e desejamos calcular a média aritmética dele:

```
#calculo da média
notas = [6,7,5,8,9]
soma=0
x=0
while x<5:
    soma += notas[x]
    x += 1
printf(f"Média: {soma/x:5.2f}")
```

```
#Cálculo da média com notas digitadas
notas = [0,0,0,0,0]
soma = 0
x=0
while x<5:
    notas[x] = float(input(f"Nota {x}: "))
    soma += notas[x]
    x +=1
x=0
while x<5:
    prinf(f"Nota{x}: {notas[x]:6.2f}")
    x += 1
printf(f"Média: {soma/x:5.2f}")
```

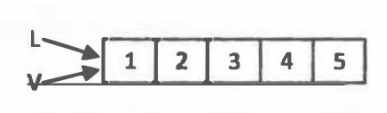
```
#Apresentação de números
numeros = [0,0,0,0,0]

x=0
while x<5:
    numeros[x] = int(input(f"Número {x+1}: "))
    x +=1
while True
    escolhido = int(input("Que posição você quer imprimir (0
    if escolhido == 0:
        break;
    print(f"Voce escolheu o número: {numeros[escolhido-1]}")
```

Uma lista em Python é um **objeto** e, quando atribuímos um objeto a outro, estamos apenas copiando a mesma referência da lista e, não seus dados em si. No caso abaixo, `V` funciona como um apelido de `L`, ou seja, `V` e `L` são a mesma lista.

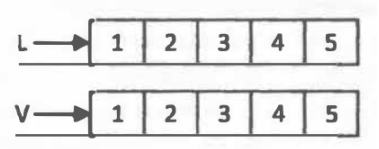
```
L = [1,2,3,4,5]
V = L
V[0]
V
L
```

Quando modificamos `V[0]`, estamos modificando o mesmo valor de `L[0]`, pois ambos são **referências**, ou **apelidos** para a mesma lista na memória.



Para criar uma cópia **independente** de uma lista, utilizaremos outra sintaxe.

```
L = [1,2,3,4,5]
V = L[:]
V[0]=6
V
L
```



Podemos fatiar uma lista, da mesma forma que fizemos com strings.

```
L = [1,2,3,4,5]
L[0:5]
L[:5]
L[:-1]
L[1:3]
L[1:4]
L[3:]
L[-2]
```

Podemos usar a função `len()` com listas.

```
#Repetição do tamanho da lista usando len()
L=[1,2,3]
x=0
while x<len(L):
    print(L[x])
    x +=1
```

Adição de elementos da lista

Para adicionar um elemento ao fim da lista, utilizaremos o método `append()`. Em Python, chamamos um método escrevendo o nome dele após o nome do objeto.

```
#Adição de elementos à Lista
L=[]
while True:
    n=int(input("Digite um número (0 sai): "))
    if n==0:
        break
    L.append(n)
x=0
while x<len(L)
    print(L[x])
    x +=1
```

Outra forma de adicionarmos a lista;

```
L=[]
L = L+[1]
L = L + [3,4,5] # o compilador executa automaticamente o
# método extend(lista[])
L.extend([6,7,8])
```

Remoção de elementos da lista

Podemos retirar alguns elementos da lista ou todos eles. Utilizaremos a instrução **del**.

```
L=["a","b","c","d","e"]
del L[1]
L
del L[1:3]
L
```

Exemplo:

```
# uma lista de Bandas
bandas = ['Ivete Sangalo', 'Wesley Safadão', 'Pink Floyd', 'A

#varrendo a lista inteira
for banda in bandas:
    print banda

# trocando o último elemento
bandas[-1] = 'GG da Bahia'

# incluindo
bandas.append('AC/DF')

#removendo
bandas.remove('Welsey Safadão')

#ordena a lista
bandas.sort()

#ordena sem alterar seus elementos
sorted(bandas)

#inverte a lista
bandas.reverse()

#Imprime numerado
print(list(enumerate(bandas)))
```

```
for i, banda in enumerate(bandas):
    print(i+1, '=>', banda)

#Imprime do segundo item em diante
print bandas[1:]
```

Enumerate

A função **enumerate()** gera uma tupla em que o primeiro valor é o índice e o segundo é o elemento da lista sendo enumerada.

```
L=[5,9,13]
x=0
for e in L:
    print(f"[{x}] {e}")
    x+=1

L = [5,9,13]
for x,e in enumerate(L):
    print(f"[{x}] {e}")

#enumerate(L) vai gera tupla (0,5), (1,9), (2,13)
```

💡 A função `enumerate()` retorna uma **tupla** de dois elementos a cada iteração: um número sequencial e um item da sequência correspondente.

💡 As operações de ordenação (***sort***) e inversão (***reverse***) são realizadas na própria lista, sendo assim, não geram novas listas.

```
#Usando listas como filas
#O primeiro a chegar é o primeiro a sair (FIFO - First In Fir
lista = ['A', 'B', 'C']
print('lista:', lista)

# A lista vazia é avaliada como falsa
while lista:

    #em filas, o primeiro item é o primeiro a sair
    # pop(0) remove e retorna o primeiro item
    print('Saiu', lista.pop(0), ', faltam', len(lista))

#Usando listas como pilhas
#O último elemento a chegar é o primeiro a sair (LIFO - Last
#Mais itens na lista
lista +=['D', 'E', 'F']
print('lista: ', lista)

while lista:

    # em pilhas, o primeiro item é o último a sair
    # pop() remove e retorna o último item
    print('Saiu', lista.pop(), ', faltam', len(lista))
```

Exemplo de fila

```
#Simulação de uma fila de banco
ultimo = 10
fila= list(range(l, ultimo + 1))
```

```
while True:
    print(f"\nExistem {len(fila)} clientes na fila")
    print(f"Fila atual: {fila}")
    print( "Digite F para adicionar um cliente ao fim da fila
    print( "ou A para realizar o atendimento. S para sair.")
    operação= input("Operação (F, A ou S):")
    if operação== "A":
        if len(fila) > 0:
            atendido= fila.pop(0)
            print(f"Cliente {atendido} atendido")
        else:
            print("Fila vazia! Ninguém para atender.")
    elif operação== "F":
        ultimo += 1 # Incrementa o ticket do novo cliente
        fila.append(ultimo)
    elif operação == "S":
        break
    else:
        print( "Operação inválida! Digite apenas F, A ou S!")
```

Exemplo de pilha

```
#Pilha de pratos
prato= 5
pilha= list(range(l, prato+ 1))
while True:
    print(f"\nExistem {len(pilha)} pratos na pilha")
    print(f"Pilha atual: {pilha}")
    print("Digite Epara empilhar um novo prato,")
    print( "ou D para desempilhar. S para sair.")
    operação= input( "Operação (E, D ou S): ")
    if operação== "D":
        if len(pilha) > 0:
            lavado= pilha.pop(-1)
```

```
            print(f"Prato {lavado} lavado")
        else:
            print("Pilha vazia! Nada para lavar.")
    elif operação== "E":
        prato+= 1 # Novo prato
        pilha.append(prato)
    elif operação == "S":
        break
    else:
        print("Operação invãlida! Digite apenas E, D ou S!")
```

Exemplo: Fazendo uma procura

```
L=[7,9,10,12]
p = int(input("Digite um número a pesquisa:"))
for e in L:
    if e==p:
        print("Elemento encontrado!")
        break
else: #parecido com o da instrução else, a instrução while po
    print("Elemento não encontrado")
```

Listas dentro de listas

Vimos que strings podem ser indexadas ou acessadas letra por letra. Um fator interessante é que podemos acessar as strings dentro da lista, letra por letra, usando um segundo índice:

```
S = ["maçãs", "peras", "kiwis"]
print(S[0][0])
```

```
print(S[0][1])
print(S[1][1])
```

Isso nos leva a outra vantagem das listas em Python: listas dentro de listas. Temos também que os elementos de uma lista não precisam ser do mesmo tipo.

```
#Listas com elementos de tipos diferentes
produto1 = ["maçã", 10, 0.30]
produto2 = ["pera", 5, 0.75]
produto3 = ["kiwi", 4, 0.98]
compras = [produto1, produto2,produto3]
print(compras)
for e in compras:
    print(f"Produto: {e[0]} ")
    print(f"Quantidade: {e[1]}")
    print(f"Preço: {e[2]:5.2f}")
```

```
#Criação e impressão da lista de compras
compras=[]
while True:
    produto = input("Produto: ")
    if produto == "fim":
        break
    quantidade = int(input("Quantidade: "))
    preco = float(input("Preço: "))
    compras.append([produto, quantidade, preço])
soma=0.0
for e in compras:
    print(f"e[0]:20s x {e[1]:5d} {e[2]:5.2f}{e[1]*e[2]:6.2f}"
    soma += e[1]*e[2]
print(f"Total: {soma:7.2f}")
```

```
#exemplo de matriz

dim = 6,12
mat={}

mat[3, 7] = 3
mat[4, 6] = 5
mat[6, 3] = 7
mat[5, 4] = 6
mat[2, 9] = 4
mat[1, 0] = 9

for lin in range(dim[0]):
    for col in range(dim[1]):
        print(mat.get((lin, col),0))
    print
```

```
# Matriz em forma de string
matriz = '''0 0 0 0 0 0 0 0 0 0 0 0
                        900000000000
                        000000000400
                        000000030000
                        000000500000
                        0 0 0 0 6 0 0 0 0 0 0 0'''

mat = {}

# Quebra a matriz em linhas
for lin, linha in enumerate(matriz.splitlines()):

    # Quebra a linha em colunas
    for col, coluna in enumerate(linha.split()):

        coluna = int(coluna)
        # Coloca a coluna no resultado,
        # se for diferente de zero
```

```
        if coluna:
            mat[lin, col] = coluna
            print(mat)

# Some um nas dimensões pois a contagem começa em zero
print('Tamanho da matriz completa:', (lin + 1) * (col + 1))
print('Tamanho da matriz esparsa:', len(mat))
```

Tuplas

Semelhantes as listas, porém são **imutáveis**: não se pode acrescentar, apagar ou fazer atribuições aos itens. Tuplas são ideais para representar listas de valores constantes. Suportam a maior parte das operações de lista, como fatiamento e indexação.

💡 **Mas tuplas não podem ter seus elementos alterados.**

```
tupla = (a,b,...,z)

dias=('domingo', 'segunda','terça','quarta','quinta','sexta',
dias='domingo', 'segunda','terça','quarta','quinta','sexta','
```

```
#Particularidade: tupla com apenas um elemento
t1 = (1,)

#tuplas vazias
t=()

#Os elementos são referenciado da mesma forma que os elemento
primeiro_elemento= tupla[0]
```

```
#Lista podem ser convertidas em tuplas
tupla = tuple(lista)

#Tuplas podem ser convertidas em listas
lista = list(tupla)
```

💡 As **tuplas** são mais eficientes do que as listas convencionais, pois consomem menos recursos computacionais (memória), por serem estruturas mais simples, tal como as *strings* imutáveis em relação às *strings* mutáveis.

```
#É possível criar tuplas que contenham objetos mutáveis
lista=[3,4]
tupla=(1,2,lista)
lista=[5,6]
print(tupla)
tupla[2]=[3,4] #erro de compilação
```

Tuplas também podem ser utilizadas para desempacotar valores, por exemplo:

```
a,b = 10,20
a
b
a,b=b,a
a,b=[10,20]
*a,b=[1,2,3,4,5]
a,*b=[1,2,3,4,5]
#*a, b, dizemos: coloque o último valor em b e os restantes em
```

```
#a, *b, dizemos: coloque o primeiro valor em a e os outros em
a,*b,c=[1,2,3,4,5]
```

Range

O range é um tipo de sequência imutável de números, sendo comumente usado para *looping* de um número específico de vezes em um comando for já que representam um intervalo. O comando **range** gera um valor contendo números inteiros sequenciais, obedecendo a sintaxe:

```
##range(início, fim,passos)

for valor in range(1,10,2):
    print(valor)
```

Conjuntos

O Python provê entre os *builtins* também: `set` **(Conjuntos).** Sequência mutável unívoca (sem repetições) não ordenada. Os dois tipos implementam operações de conjuntos, tais como: união, interseção e diferença.

```
frutas = {'laranja', 'banana','uva','pera','laranja'}
frutas.add('manga')
a = set('abacate')
b = set('abacaxi')
#diferença
a-b
#uniao
a|b
#interseção
a&b
```

```
#diferença simétrica
a^b
```

```
#conjunto de dados
s1 = set(range(3))
s2 = set(range(10,7,-1))
s3 = set(range(2,10,2))

#exibe os dados
print('s1:', s1, '\ns2: ', s2, '\ns3: ',s3)

#união
s1s2 = s1.union(s2)
print('União de s1 e s2: ', s1s2)

#diferença
print('Diferença com s3: ', s1s2.difference(s3))

#intersecção
print('Intersecção ocm s3: ', s1s2.intersection(s3))

#testa se um set inclui outro
if s1.issuperset([1,2]):
    print('s1 inclui 1 e 2')

#testa se não existe elementos em comum
if s1.isdisjoint(s2):
    print('s1 e s2 não tem elementos em comum')
```

Dicionários

Um **dicionário** é uma lista de associações compostas por uma chave única e estruturas correspondentes. Dicionários são mutáveis, tais como as listas.

A chave precisa ser de um tipo imutável, geralmente são usadas ***strings***, mas também podem ser **tuplas** ou tipos numéricos. Os dicionários pertencem ao tipo de mapeamento integrado e não sequenciais como as **listas**, **tuplas** e ***strings***.

Os dicionários são delimitados por chaves (`{}`) e suas chaves ('nome', 'idade' e 'cidade') por aspas. Já os valores podem ser de qualquer tipo.

```
dicionario = {'a':a, 'b':b, ..., 'z:',z}

pessoa={'nome':'João','idade':25,'cidade': 'São Paulo'}
pessoa['Nome']
pessoa['idade']
pessoa['país']='Brasil'

a=dict(um=1,dois=2,três=3)
```

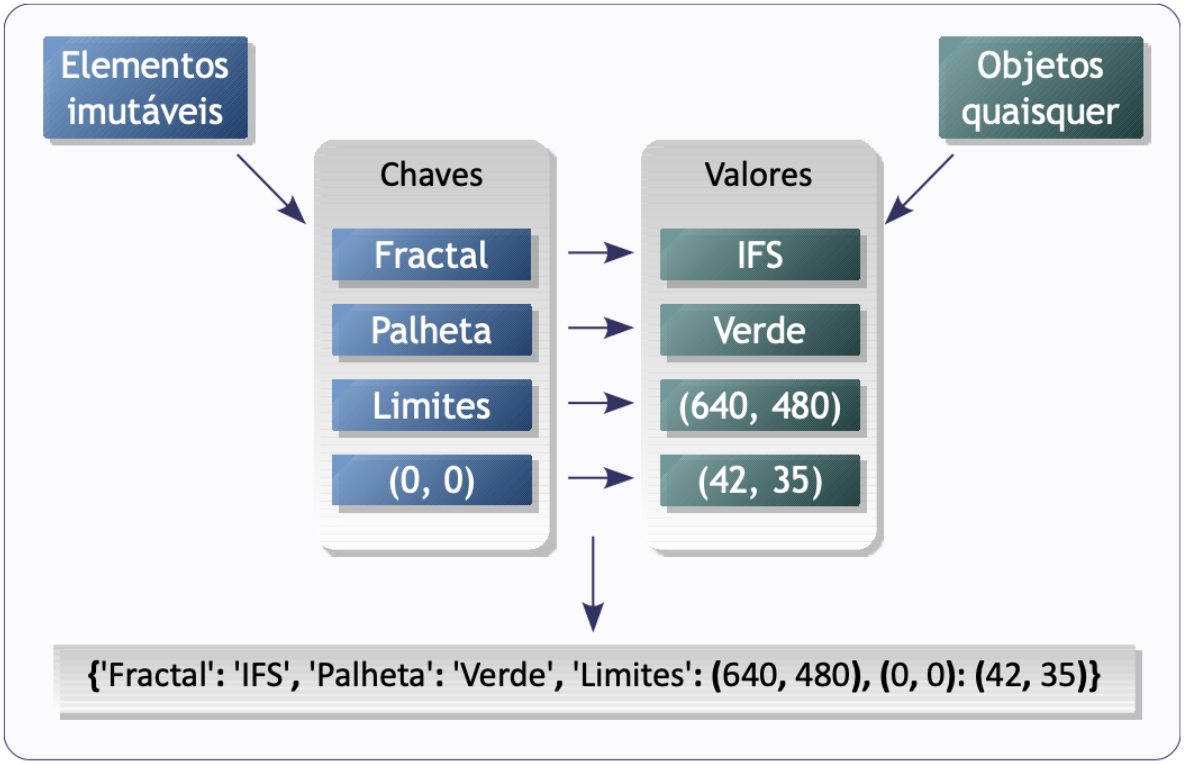
```
tabela = {"Alface": 0.45,
                    "Batata": 1.20,
                    "Tomate": 2.30,
                    "Feijão": 1.50}

print(tabela["Tomate"])

tabela["Tomate"]=2.50
print(tabela["Tomate"])

tabela["Cebola"]=2.50
print(tabela)

print("Manga" in tabela)
print("Batata" in tabela)
```




```
#exemplo de dicionário
dic = {'nome': 'Billie Joe', 'Banda':'Green Day'}

#acessando elementos
print(dic['nome'])

#adicionando elementos
dic['album'] = 'American Idiot'

#apagando um elemento do dicionário
del dic['album']

#tipos
itens = dic.items()
chaves = dic.keys()
valores=dic.values()
```

```
#bandas e seus albuns
bandas={'Green Day':['Dookie', 'American Idiot'],
                'AC/DC': ['Black Ice','Rock or Burst'],
                'Pink Floyd':['The Wall','The Dark Side of th

#mais bandas
bandas['Foo Fighters'] = ['One by One','In Your Honor']

#items() retorna uma lista de tuplas com a chave e o valor
for banda, albuns in bandas.items()
    print(banda, '=>', albuns)

#se tiver AC/DC deleta
if bandas.has_key('AC/DC'):
    del bandas['AC/DC']
```

Dicionários com Listas

Em Python, podemos ter dicionários nos quais as chaves são associadas a listas ou mesmo a outros dicionários.

```
#a chave é o nome do produto e a lista contém a quantidade e
estoque={"tomate": [1000,2.30],
                "alface": [500,0.45],
                "batata": [2001,1.20],
                "feijão": [100, 1.50]}

venda = [["tomate",5],["batata",10],["alface",5]]
total=0
print("Vendas:\n")
for operacao in venda:
    produto, quantidade = operacao
    preco = estoque[produto][1]
    custo = preco*quantidade
```

```
    print(f"{produto:12s}:{quantidade:3d}x{preco:6.2f}={custo
    estoque[produto][0]-=quantidade
    total +=custo
print(f"Custo total: {total:21.2f}\n")
print("Estoque:\n")
for chave, dados in estoque.items():
    print("Descrição: ", chave)
    print("Quantidade: ", dados[0])
    print(f"Preço: {dados[1]:6.2f}\n")
```

```
#Exemplo de dicionário sem valor padrão
d={}
for letra in "abracadabra":
    if letra in d:
        d[letra] = d[letra] +1
    else:
        d[letra] =1

print(d)

#outra forma
d={}
for letra in "abracadabra"
    d[letra] = d.get(letra,0)+1
print(d)
```

O método `get` tenta obter a chave procurada; caso não a encontre, retorna o segundo parâmetro, no caso 0 (zero). Se o segundo parâmetro não for especificado, get retornará **None.**

Observações

Em Python, o tipo booleano (*bool*) é uma especialização do tipo inteiro (*int*). O verdadeiro é chamado *True* e é igual a 1, enquanto o falso é chamado *False* e é

igual a zero.

Os seguintes valores são considerados falsos:

- *False* (falso).
- *None* (nulo).
- 0 (zero).
- '' (*string* vazia).
- [] (lista vazia).
- () (tupla vazia).
- {} (dicionário vazio).

Outras estruturas com o tamanho igual a zero.

> 💡 Além dos operadores booleanos, existem as funções `all()`, que retorna verdadeiro quando todos os itens forem verdadeiros na sequência usada como parâmetro, e `any()`, que retorna verdadeiro se algum item o for.

Resumo das estruturas de dados

	Listas	Tuplas	Dicionários	Conjuntos
Ordem dos elementos	Fixa	Fixa	Mantida a partir do Python 3.7	Indeterminada
Tamanho	Variável	Fixo	Variável	Variável
Elementos repetidos	Sim	Sim	Pode repetir valores, mas as chaves devem ser únicas	Não
Pesquisa	Sequencial, índice numérico	Sequencial, índice numérico	Direta por chave	Direta por valor
Alterações	Sim	Não	Sim	Sim
Uso primário	Sequências	Sequências constantes	Dados indexados por chave	Verificação de unicidade, operações com conjuntos

Exercícios da Aula

1. Vamos tentar resolver alguns desafios. Dada a lista = [12, -2, 4, 8, 29, 45, 78, 36, -17, 2, 12, 8, 3, 3,-52] faça um programa que:
   a. imprima o maior elemento;
   b. imprima o menor elemento;
   c. imprima os números pares;
   d. imprima o número de ocorrências do primeiro elemento da lista;
   e. imprima a média dos elementos;
   f. imprima a soma dos elementos de valor negativo
2. Faça um jogo da Forca utilizando listas. Dada uma palavra, dê algumas chances para o usuário acertar.
3. Faça um programa que leia uma expressão com parênteses. Usando pilhas, verifique se os parênteses foram abertos e fechados na ordem correta.
   Exemplo:

   `(())` **OK**

   `()()(()())` **OK**

   `( ) )` Erro

Você pode adicionar elementos à pilha sempre que encontrar abre parênteses e desempilhá-la a cada fecha parênteses. Ao desempilhar, verifique se o topo da pilha é um abre parênteses. Se a expressão estiver correta, sua pilha estará vazia no final.

4. A lista de temperaturas de Mons, na Bélgica, foi armazenada na lista T= [-10, -8, 0, 1, 2, 5, -2,-4]. Faça um programa que imprima a menor e a maior temperatura, assim como a temperatura média.
5. Escreva um programa que copie os valores pares para uma lista e os valores ímpares para outra lista. A lista inicialmente de valores é `V=[9,8,7,12,0,13,21]` .
6. Escreva um programa que controla a utilização das salas de um cinema. Imagine que a lista lugares_vagos=[10,2,1,3,0] contenha o número de lugares vagos nas salas 1,2,3,4 e 5, respectivamente. Esse programa lerá o número da sala e a quantidade de lugares solicitados. Ele deve informar se é possível vender o número de lugares solicitados, ou seja, se ainda há lugares livres. Caso seja, possível vender os bilhetes, atualizará o número de lugares livres. A saída ocorre quando se digita 0 no número da sala.
7. Escreva um programa que gere um dicionário, em que cada chave seja um caractere, e seu valor seja o número desse caractere encontrado em uma frase lida.

   Exemplo:O rato → {"O":1,"r":1,"a":1,"t":1,"o":1}
8. Escreva um programa que compare duas listas. Utilizando operações com conjuntos, imprima:

- os valores comuns às duas listas
- os valores que só existem na primeira
- os valores que existem apenas na segunda
- uma lista com os elementos não repetidos das duas listas.
- a primeira lista sem os elementos repetidos na segunda

9. Escreva um programa que compare duas listas. Considere a primeira lista como a versão inicial e a segunda como a versão após alterações. Utilizando operações com conjuntos, seu programa deverá imprimir a lista

de modificações entre essas duas versões, listando:
    - os elementos que não mudaram
    - os novos elementos
    - os elementos que foram removidos

10. Use um dicionário para armazenar informações sobre uma pessoa que você conheça. Armazene seu primeiro nome, o sobrenome, a idade e a cidade em que ela vive. Você deverá ter chaves como first_name, last_name, age e city. Mostre cada informação armazenada em seu dicionário.
11. Comece com o programa que você escreveu no Exercício 10. Crie dois novos dicionários que representem pessoas diferentes e armazene os três dicionários em uma lista chamada people. Percorra sua lista de pessoas com um laço. À medida que percorrer a lista, apresente tudo que você sabe sobre cada pessoa.
12. Crie vários dicionários, em que o nome de cada dicionário seja o nome de um animal de estimação. Em cada dicionário, inclua o tipo do animal e o nome do dono. Armazene esses dicionários em uma lista chamada pets. Em seguida, percorra sua lista com um laço e, à medida que fizer isso, apresente tudo que você sabe sobre cada animal de estimação.
13. Crie uma lista chamada sandwich_orders e a preencha com os nomes de vários sanduíches. Em seguida, crie uma lista vazia chamada finished_sandwiches. Percorra a lista de pedidos de sanduíches com um laço e mostre uma mensagem para cada pedido, por exemplo, Preparei seu sanduíche de atum. À medida que cada sanduíche for preparado, transfira-o para a lista de sanduíches prontos. Depois que todos os sanduíches estiverem prontos, mostre uma mensagem que liste cada sanduíche preparado.
14. Escreva um jogo da velha para dois jogadores. O jogo deve perguntar onde você quer jogar e alternar entre os jogadores. A cada jogada, verifique se a posição está livre. Verifique também quando um jogador venceu a partida. Um jogo da velha pode ser visto como uma lista de 3 elementos, na qual cada elemento é outra lista também com três elementos.