

# Agrupando objetos

# Principais conceitos a serem abordados

- Listas
- Coleções
- Loops

# O requisito para agrupar objetos

- Várias aplicações envolvem coleções de objetos:
  - agendas pessoais;
  - catálogos de bibliotecas; e
  - sistema de registro de alunos.
- O número de itens a serem armazenados varia.
  - Itens adicionados.
  - Itens excluídos.

# O projeto “*weblog-analyzer*”

- Um servidor Web registra os detalhes de cada acesso.
- Suporte a tarefas do webmaster.
  - Páginas mais populares.
  - Períodos mais ocupados.
  - Quantos dados são entregues.
  - Referências quebradas.
- Analisa os acessos por hora.
  - 2006 06 07 03 45      07/06/2006 03:45<sub>4</sub>

# Criando um objeto com list

```
class LogAnalyzer:
```

```
    def __init__(self):
```

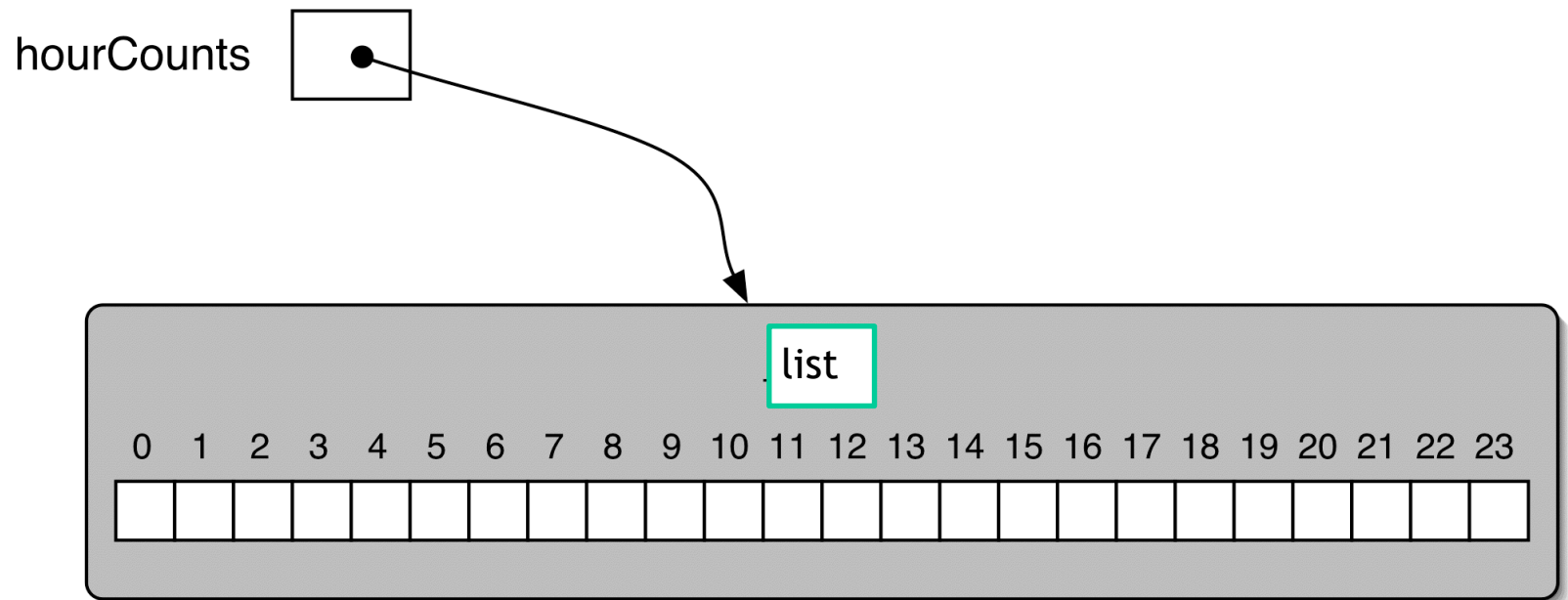
```
        self.__hourCounts = list()
```

```
    ...
```

Criação do objeto  
list



# A lista \_\_hourCounts



# Utilizando uma lista

- A notação entre colchetes é utilizada para acessar um elemento da lista:  
`self.__hourCounts[...]`
- Elementos são utilizados como variáveis convencionais.
  - À esquerda de uma atribuição:
    - `self.__hourCounts[hour] = ...`
  - Em uma expressão:
    - `adjusted = self.__hourCounts[hour] - 3`
    - `x = self.__hourCounts[hour] + 1`

# Um bloco de notas pessoais

- Notas podem ser armazenadas.
- Notas individuais podem ser visualizadas.
- Não há um limite para o número de notas.
- Haverá informação de quantas notas estão armazenadas.



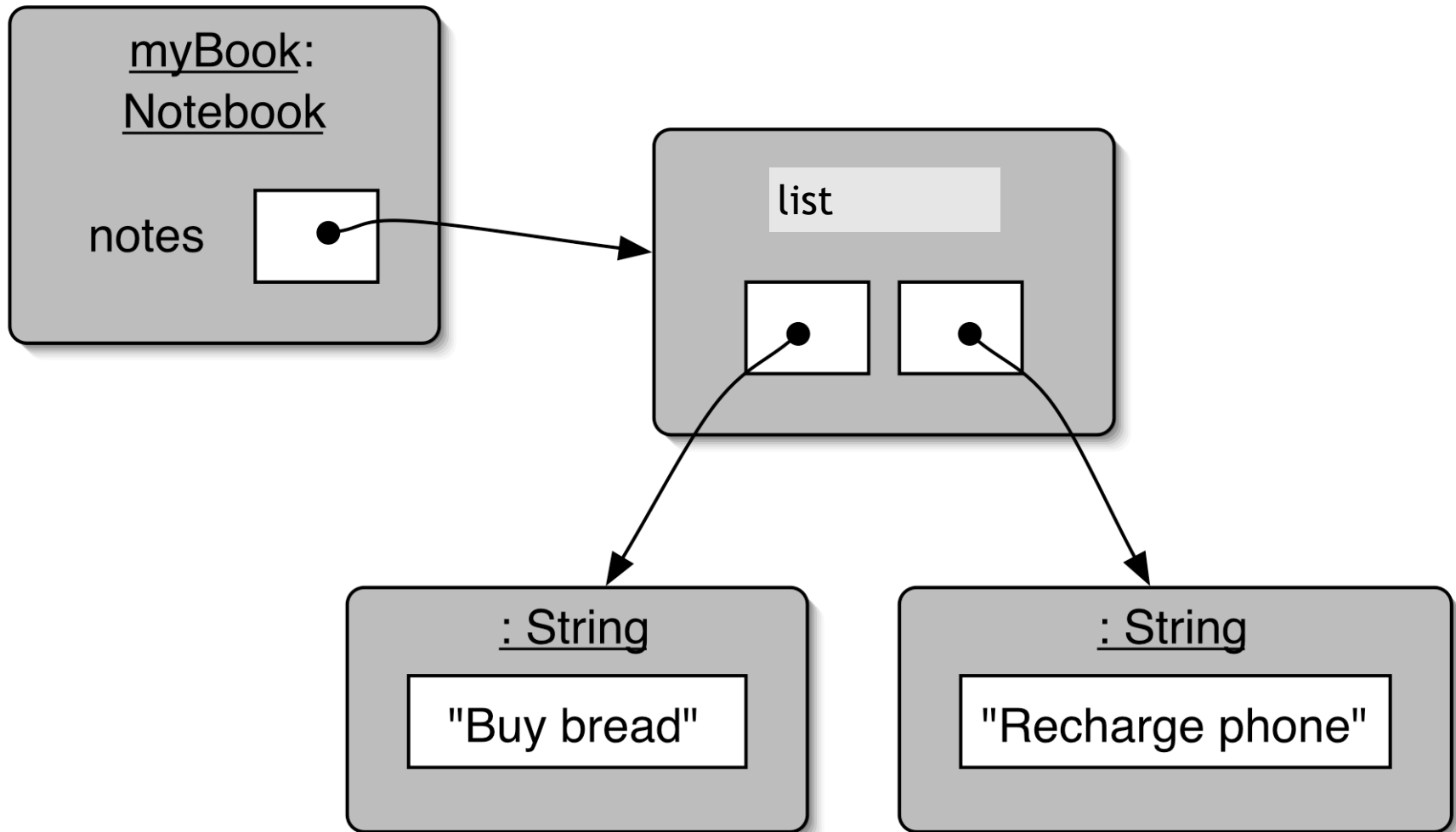
```
class Notebook:

    def __init__(self):

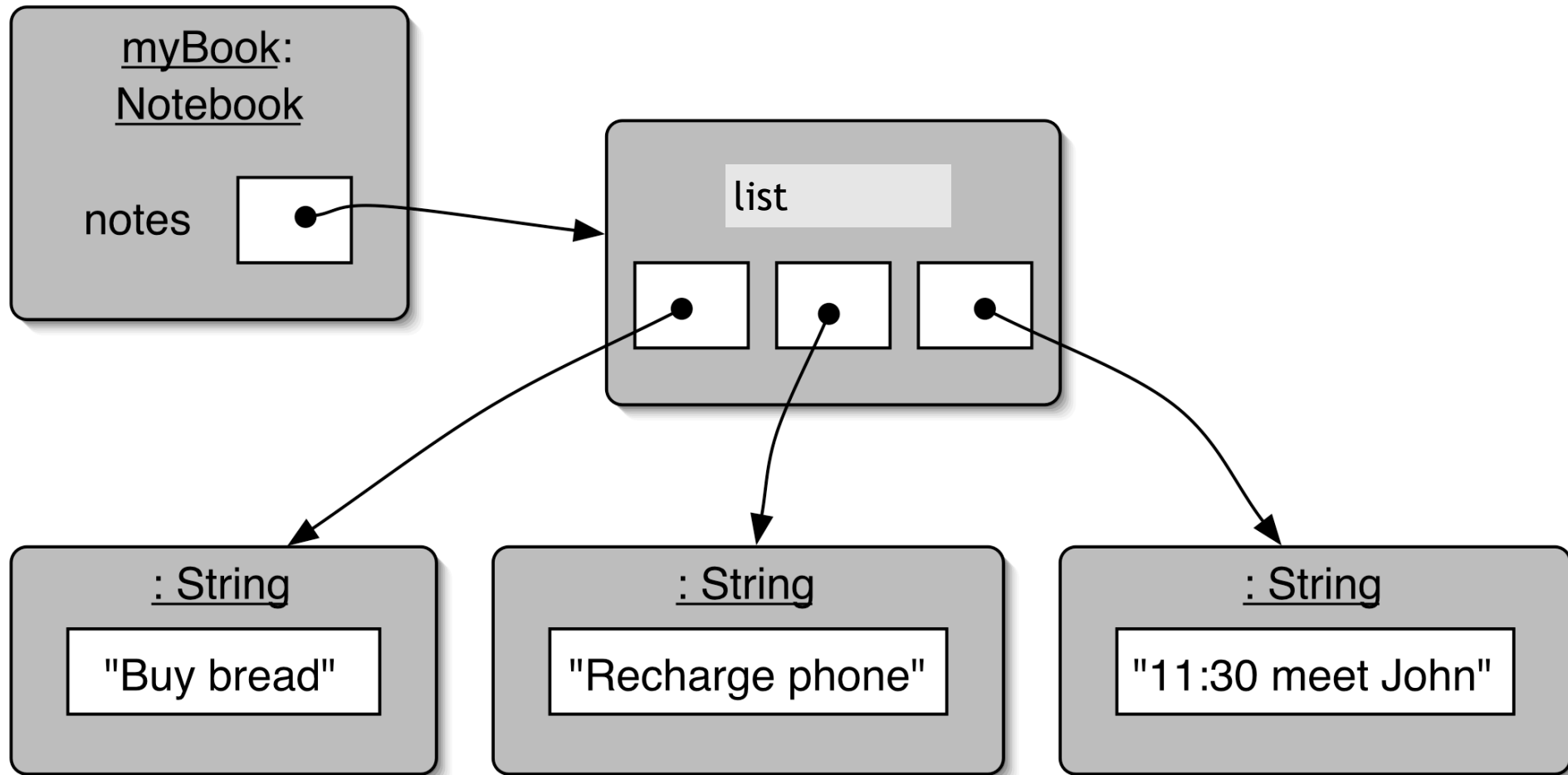
        self.__notes = list()

    ...
```

# Estruturas de objetos com coleções



# Adicionando uma terceira nota



# Recursos de list

- A capacidade interna é aumentada conforme necessário.
- Mantém uma contagem privada.
- Mantém os objetos em ordem.
- Detalhes sobre como tudo isso é feito são ocultos.
  - Isso importa?
  - Não saber como é feito nos impede de utilizá-los?

# Utilizando uma lista


```
class Notebook:
```

```
...
```

```
def storeNote(self, note)
```

```
    self.__notes.append(note)
```


Adicionando uma  
nova nota



```
def numberOfNotes(self):
```

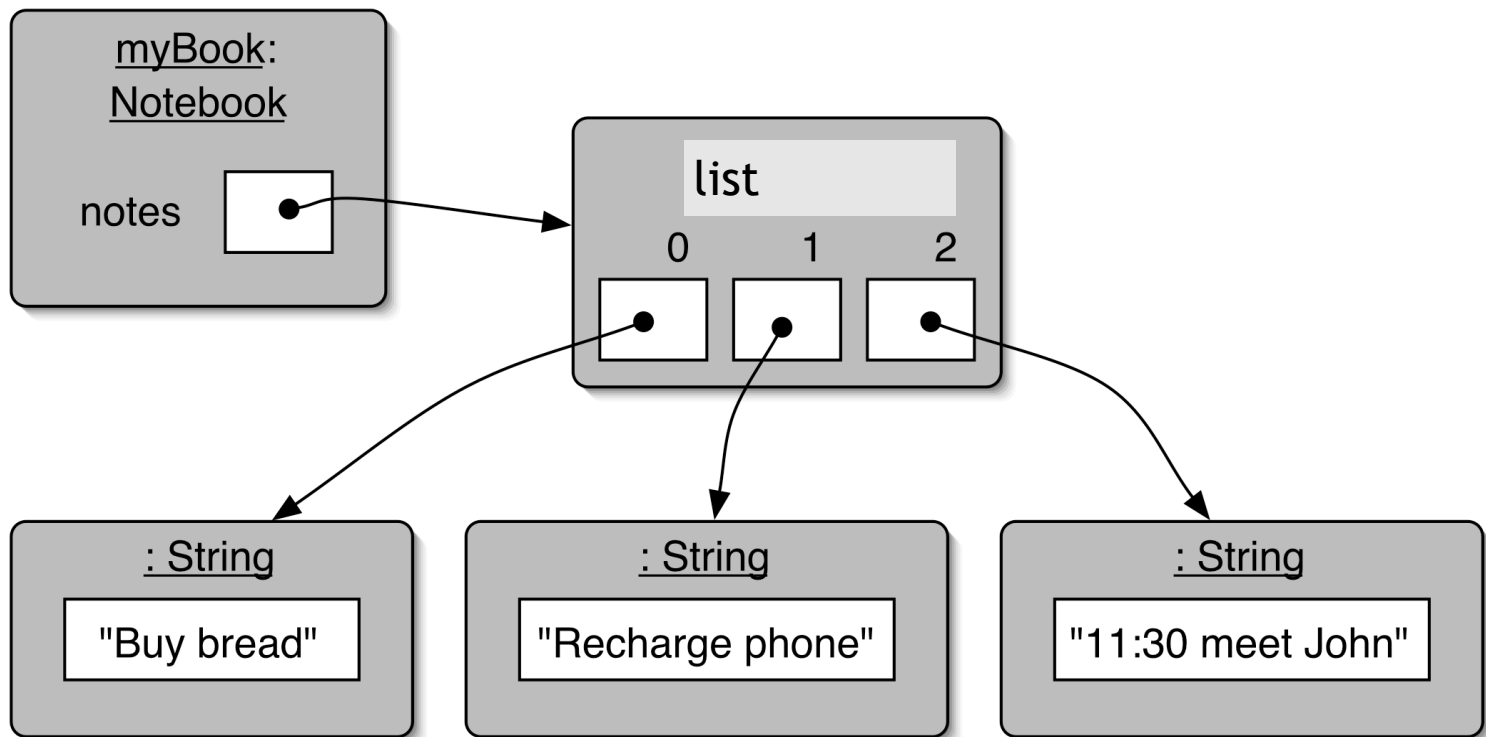
```
    return len(self.__notes)
```

Retornando o número  
de notas



```
...
```

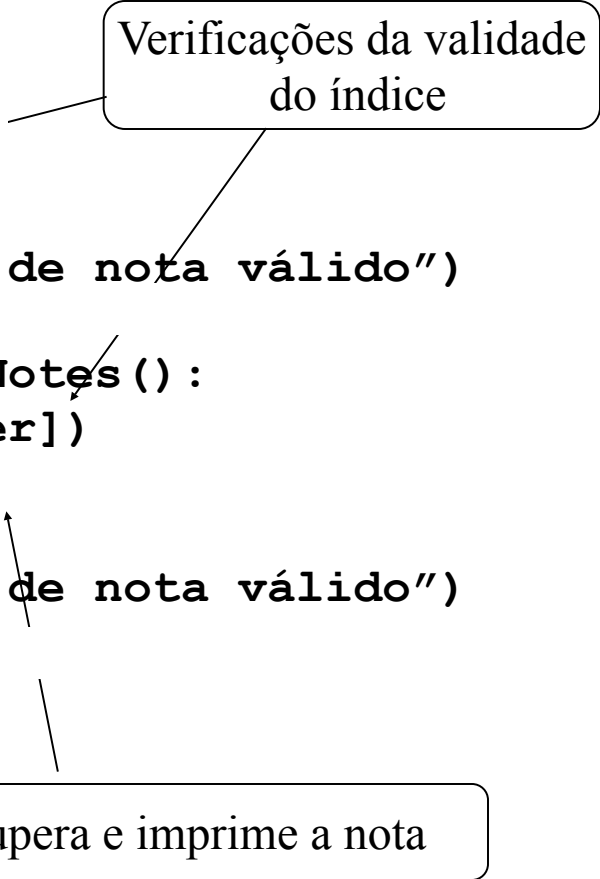
# Numeração de índice



# Recuperando um objeto

```
def showNote(self,noteNumber):  
  
    if noteNumber < 0:  
        print("Este não é um número de nota válido")  
  
    elif noteNumber < self.numberOfNotes():  
        print(self.__notes[noteNumber])  
  
    else :  
        print("Este não é um número de nota válido")
```

Verificações da validade  
do índice



Recupera e imprime a nota

# Removendo um objeto

```
def removeNote(self, note)

    if note in self.__notes:
        self.__notes.remove(note)

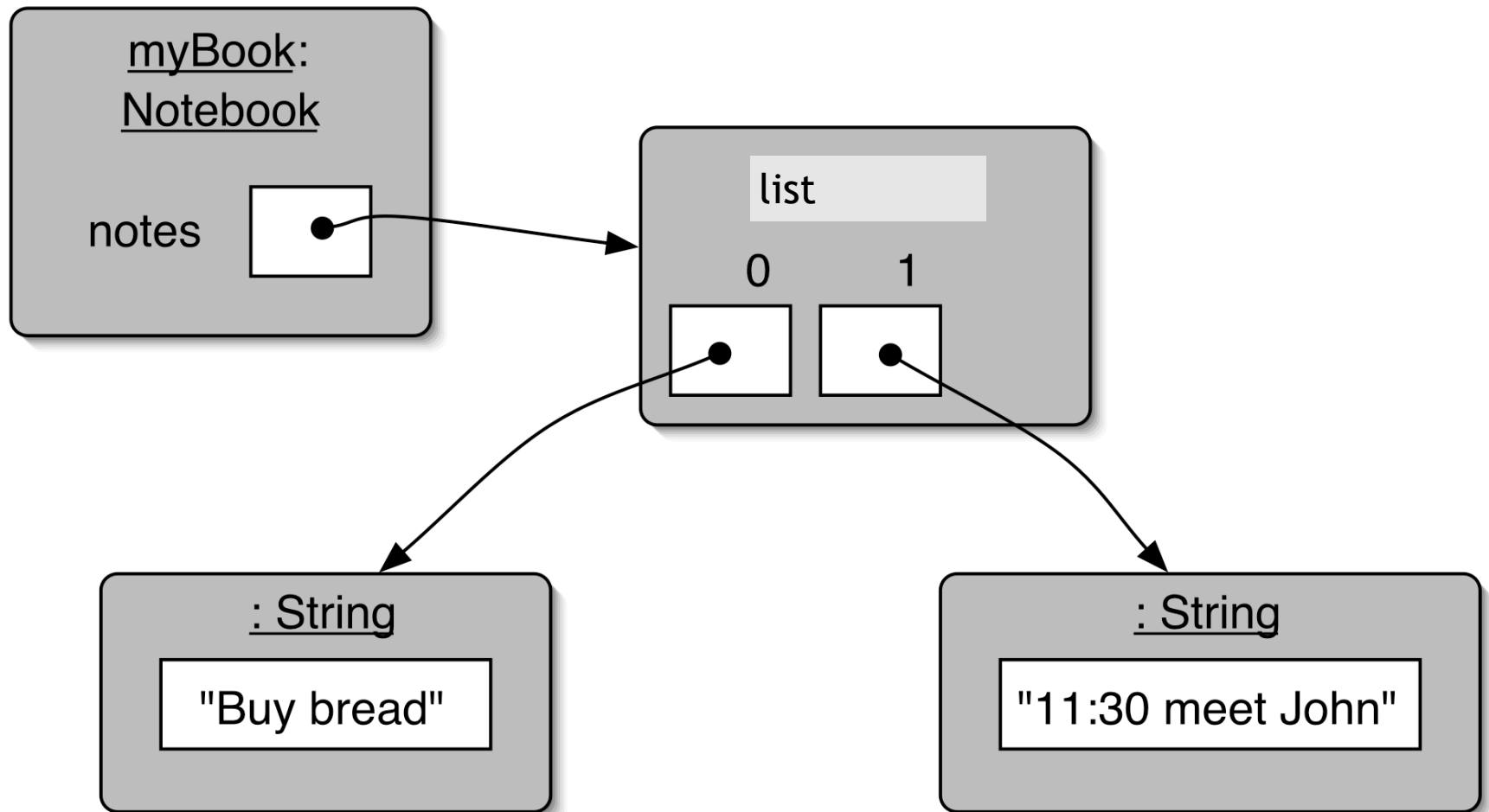
    else:
        print("Esta não é uma nota válida")
```



Remove a nota



# Remoção pode afetar a numeração



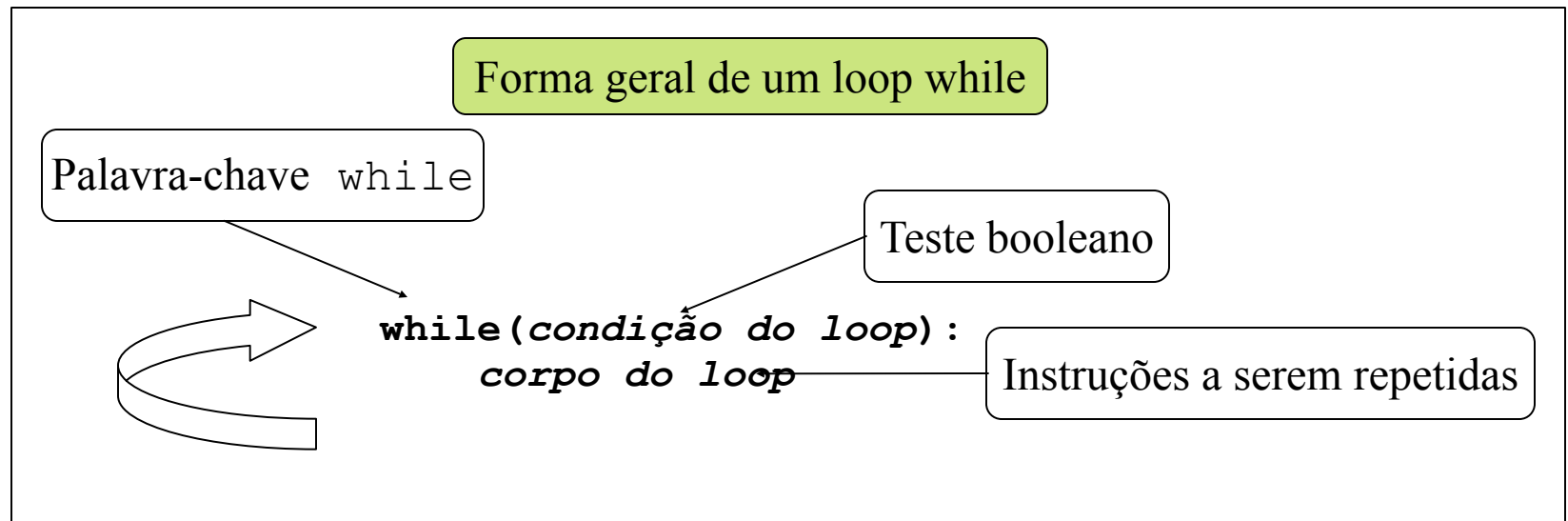
# Iteração

- Frequentemente, queremos realizar algumas ações em um número arbitrário de vezes.
  - Por exemplo, imprimir todas as notas na agenda. Quantas existem?
- A maioria das linguagens de programação inclui instruções de *loop* para tornar isso possível.
- Python tem duas categorias de instruções de loop (*while* e *for*).
  - Vamos começar com o loop *while*.

# O loop `while`

- Sintaxe semelhante à linguagem C
- Utilizado frequentemente para iterar por um número arbitrário de vezes.
- Utilizado para iterar por uma lista.

# Pseudocódigo do loop `while`



Exemplo de pseudocódigo para imprimir  
todas os itens

```
while(há pelo menos mais um item a ser impresso):  
    mostre o próximo item
```

# 0 loop while

---

```
# Soma de 0 a 99
soma=0
contador=1
while contador < 100:
    soma=soma+contador
    contador=contador + 1
print (soma)
```

# Um exemplo Python

```
"""
```

```
    Lista todas as notas no bloco de notas.
```

```
"""
```

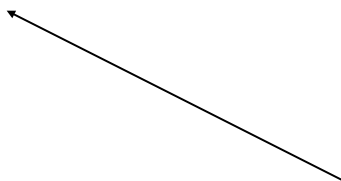
```
def listNotes(self):
```

```
    index = 0
```

```
    while index < numberOfNotes():
```

```
        print(self.__notes[index])
```

```
        index += 1
```



Incrementa por um

# Iterando com for

- O loop **for** trabalha com list da seguinte maneira:

```
for VARIABLE in LIST:  
    BODY
```

# Iterando com for

- exemplo:

```
friends = ["Joe", "Zoe", "Adam", "Josy", "Zuki", "Gray"]  
for friend in friends:  
    print(friend)
```



# Iterando com for

---

## ● Sintaxe:

for <referência> in <sequência>:

    <bloco de código>

else:

    <bloco de código>

# Iterando com for

---

- # Soma de 0 a 99
- soma=0
- for contador in range(1, 100):  
    soma=soma + contador
- print (soma)

# Iterando com for

---

- A função `range(m, n, p)`, é muito útil em laços,
- pois retorna uma lista de inteiros, começando em `m` e menores que `n`, em passos de comprimento `p`,
- que podem ser usados como sequência para o laço.

# Métodos de list

- exemplos:

```
>>> mylist = [ ]
```

```
>>> mylist.append(5)
```

```
>>> mylist.append(27)
```

```
>>> mylist.append(3)
```

```
>>> mylist.append(12)
```

```
>>> mylist
```

```
[5, 27, 3, 12]
```

# Métodos de list

- exemplos:

```
>>> mylist.insert(1, 12) # Insere 12 na pos 1, desloca  
    outros itens para o final
```

```
>>> mylist
```

```
[5, 12, 27, 3, 12]
```

```
>>> mylist.count(12) # quantas vezes 12 em mylist?
```

```
2
```

# Métodos de list

- exemplos:

```
>>> mylist.extend([5, 9, 5, 11]) # coloca toda lista no  
    final de mylist
```

```
>>> mylist
```

```
[5, 12, 27, 3, 12, 5, 9, 5, 11]
```

```
>>> mylist.index(9) # encontra index do primeiro 9  
    em mylist
```

```
6
```

# Métodos de list

- exemplos:

```
>>> mylist.reverse()
```

```
>>> mylist
```

```
[11, 5, 9, 5, 12, 3, 27, 12, 5]
```

```
>>> mylist.sort()
```

```
>>> mylist
```

```
[3, 5, 5, 5, 9, 11, 12, 12, 27]
```

# Métodos de list

- exemplo:

```
>>> mylist.remove(12) # Remove o primeiro 12 na  
    lista
```

```
>>> mylist
```

```
[3, 5, 5, 5, 9, 11, 12, 27]
```



# Revisão (1)

- Coleções permitem que um número arbitrário de objetos seja armazenado.
- Bibliotecas de classes normalmente contêm classes da coleção, experimentadas e testadas.
- Bibliotecas de classe Python são chamadas *pacotes*.

## Revisão (2)

- Itens podem ser adicionados e removidos.
- Todo item tem um índice.
- Valores de índice podem mudar se os itens forem removidos (ou, então, se outros itens forem adicionados).

# Revisão (3)

- Instruções de loop permitem que um bloco de instruções seja repetido.
- Um loop `while` Python permite que a repetição seja controlada por uma expressão booleana.
- Um loop `for` Python permite que a iteração seja feita em uma coleção.