Python Help

PUBLICADO POR

VALDIR STUMM JR
POSTADO NO

25 DE SETEMBRO DE 2012
PUBLICADO EM

ESTRUTURAS, LISTAS
COMENTÁRIOS

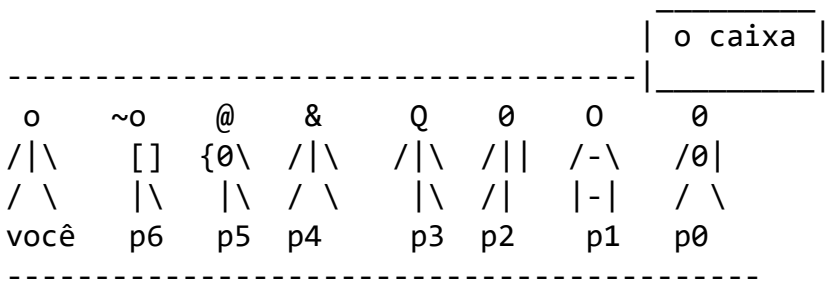
14 COMENTÁRIOS

Filas e Pilhas em Python

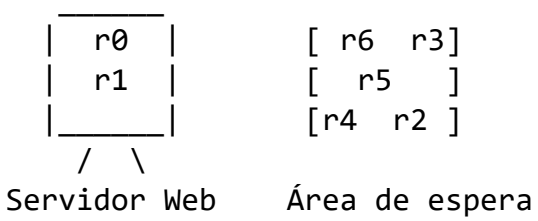
Você pode não perceber, mas boa parte dos programas que você usa no dia-a-dia utilizam internamente os tipos abstratos de dados chamados filas e pilhas. Eles são chamados dessa forma porque representam uma estrutura de dados com operações associadas que definem um certo comportamento. Eles são muito úteis para o programador pois podem ser utilizados em diversas situações e têm um *grande poder*de simplificar algoritmos. A seguir, explicarei cada uma dessas estruturas e demonstrarei como poderíamos implementá-las em Python.

Filas

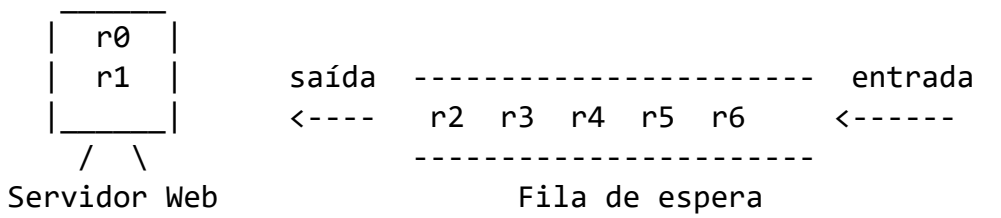
Tenho certeza que você já sabe como uma fila funciona. Pense na fila que você pega toda vez que vai ao supermercado. Para que você seja atendido, é preciso que todas as pessoas que estiverem na sua frente na fila sejam atendidas ou desistam para que você seja atendido, certo? Por isso, podemos dizer que a idéia da fila é que o último a entrar na fila seja o último a sair.



No exemplo acima, é preciso que p0, p1, p2, p3, p4, p5 e p6 sejam atendidas para que seja a sua vez de passar as compras e pagar por elas. Uma estrutura de dados **Fila** funciona da mesma maneira. Uma fila é uma estrutura utilizada especificamente para armazenamento temporário de dados na memória. Diferentemente de uma lista, que também serve para esse fim, uma fila possui regras quanto a qual elemento será retirado e onde será inserido um novo elemento. Em uma lista, é possível inserir elementos em qualquer posição e retirar quaisquer elementos, não importando sua posição. **Em uma fila**, os novos elementos que são inseridos são sempre posicionados ao final dela. Quando se faz a retirada de um elemento de uma fila, sempre iremos retirar o elemento que há mais tempo está nela (o primeiro). Imagine um servidor Web (https://pt.wikipedia.org/wiki/Servidor_web)que é capaz de atender a, no máximo, 2 requisições simultâneas. O que acontece quando chega uma terceira requisição no momento em que o servidor já está ocupado atendendo a 2 requisições? A solução mais óbvia seria colocar a nova requisição em uma área de espera. Assim, quando o servidor terminar de atender uma das atuais 2 requisições, ele poderá retirar a nova requisição da área de espera e atendê-la. E se, enquanto o servidor estiver ocupado atendendo a 2 requisições, chegarem em sequência 5 novas requisições?



O servidor poderia colocar as novas requisições em uma área de espera, e sempre que houver disponibilidade, retirar uma de lá e processá-la, até que não hajam mais requisições a serem processadas. A solução é boa, mas para ser justo com as solicitações que chegam dos usuários, o servidor deve processar primeiro as que já estão esperando há mais tempo, correto? Assim, o melhor a se fazer é utilizar uma **fila**para armazenar as requisições que estão em espera.



Dessa forma, a requisição que está há mais tempo esperando é a primeira a sair da fila quando o servidor tiver disponibilidade.

Implementando uma fila

Uma fila nada mais é do que uma estrutura de armazenamento com políticas de ordem de entrada e saída de elementos. Assim, ela pode ser implementada usando uma lista, por exemplo. Como poderíamos fazer? Sabemos que as listas oferecem alguns métodos conhecidos para inserção/remoção de elementos:

- `.append(elemento)`: adiciona `elemento` ao final da lista;
- `.insert(índice, elemento)`: insere `elemento` após a posição `índice`;
- `.pop(índice)`: remove e retorna o elemento contido na posição `índice`.

Para inserir elementos, podemos usar o método `append()`, que insere um elemento ao final de uma lista. Para a retirada de elementos, podemos utilizar o método `pop(x)`, que retira e retorna um elemento da posição `x`. Em se tratando de uma fila em que estamos inserindo elementos no final, de qual posição devemos retirar os elementos? Acertou quem pensou *"da primeira"*. Isso mesmo vamos remover o primeiro elemento utilizando `pop(0)`. Veja:

```
>>> fila = [10, 20, 30, 40, 50]
>>> fila.append(60)  # insere um elemento no final da fila
>>> print fila
[10, 20, 30, 40, 50, 60]
>>> print fila.pop(0) # remove o primeiro elemento da lista
10
>>> print fila
[20, 30, 40, 50, 60]
>>> print fila.pop(0) # remove o primeiro elemento da lista
20
>>> print fila
[30, 40, 50, 60]
>>> print fila.pop(0) # remove o primeiro elemento da lista
30
>>> print fila
[40, 50, 60]
```

Você deve estar se perguntando: *"e se eu fizesse o contrário, inserindo no início e removendo do final, funcionaria também como uma fila?"* Sim, funcionaria. Pois, da mesma forma, teríamos a política do **primeiro a entrar, primeiro a sair** também conhecida como *First-In, First-Out*, ou simplesmente FIFO. Agora, para criar uma forma consistente de usar filas em Python, vamos criar uma classe para encapsular as operações da fila. Podemos definir uma classe `Fila`que internamente tenha uma lista representando o local onde serão armazenados os elementos. Essa classe deverá ter dois métodos principais:

- `insere(elemento)`: recebe como entrada um elemento a ser inserido na fila.
- `retira()`: remove um elemento da fila e o retorna para o chamador.

Complete a implementação de fila abaixo, de acordo com o que foi explicado acima. O código completo se encontra no final deste texto, para que você possa comparar.

```
class Fila(object):
    def __init__(self):
        self.dados = []

    def insere(self, elemento):
        ____________________________________

    def retira(self):
        ____________________________________
```

Tendo a classe acima definida em um arquivo `fila.py`, poderíamos importar o módulo e usar tal classe:

```
>>> import fila
>>> f = fila.Fila()
>>> f.insere(10)
>>> f.insere(20)
>>> f.insere(30)
>>> print f.retira()
10
>>> print f.retira()
20
```

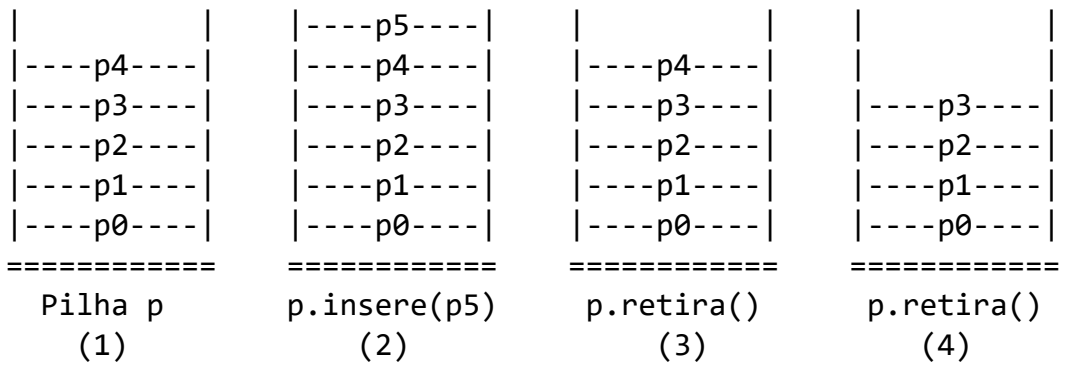
Importante: segundo a própria documentação oficial Python, uma lista não é a estrutura mais recomendada para a implementação de uma fila, porque a inserção ou remoção de elementos do início da lista é lenta, se comparada à inserção/remoção do final da lista. Assim, é sugerido que seja usado um `deque` (`collections.deque` (http://docs.python.org/library/collections.html#collections.deque)).

Só isso?

Não, nós vimos apenas as filas "comuns". Voltando ao exemplo da fila do supermercado, imagine que você está chegando aos caixas e ainda não decidiu qual deles vai usar. Aí você vê que o caixa de atendimento preferencial para idosos/gestantes/deficientes possui apenas 3 pessoas na fila, enquanto que os outros caixas têm filas com no mínimo 10 pessoas. Como o caixa não é exclusivo para idosos, você vai lá e entra na fila. Outras pessoas percebem a sua "sagacidade" (:P) e fazem o mesmo. Fica tudo certo até que chega um idoso para entrar na fila e todas as pessoas que estão na fila são obrigados a ceder sua vez para o idoso. E eis que chega o clube da terceira idade inteiro para ser atendido logo em seguida. O que acontece? As pessoas que estão na fila cedem sua vez para os idosos, e assim, seu atendimento vai sendo postergado cada vez mais. Enfim, esse é um exemplo de uma *fila de prioridades*. Elementos (as pessoas) que possuem maior prioridade, saem antes da fila. Se quiser saber mais, veja aqui (http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/priority.html).

Pilhas

Assim como as filas, aposto que você sabe exatamente como funciona uma pilha. Pense em uma pilha de papéis sobre uma mesa. Se você precisar adicionar um papel a essa pilha, você o adiciona sobre a pilha, ou seja, no topo dela, certo? E quando vai retirar um papel da pilha, você começa pelo que estiver no topo, certo? Ou seja, diferentemente da fila (FIFO), em uma pilha o último a entrar nela, é o primeiro a sair (Last-In, First-Out — LIFO). Veja abaixo uma ilustração de como funciona uma pilha. No primeiro item da ilustração (1), temos uma pilha composta por 5 elementos, que foram inseridos cronologicamente na seguinte ordem: p0, p1, p2, p3, e, por fim, p4. A segunda parte da ilustração (2) mostra o estado da pilha p após ter sido realizada a inserção de um novo elemento (p5). A terceira parte (3) mostra a pilha p após haver a retirada de um elemento. Como você pode ver, o elemento retirado foi o último a ter sido inserido (p5). Na última parte da ilustração (4), é apresentada mais uma retirada de elemento da pilha, desta vez p4. Se houvessem mais operações de retirada, teríamos a retirada, em ordem, dos elementos: p3, p2, p1 e p0.



Tranquilo, né? O que você deve lembrar sobre as pilhas é que ela usa uma política LIFO (Last-In, First-Out) para inserção/retirada de elementos.

Onde as pilhas são usadas?

Talvez a mais famosa utilização de pilhas em computação esteja no gerenciamento de chamadas de função de um programa. Uma pilha pode ser usada para manter informações sobre as funções de um programa que estejam ativas, aguardando por serem terminadas. Considere o seguinte exemplo:

```python
def ola():
    print "olá, "
    mundo()

def mundo():
    print "mundo!"

def olamundo():
    ola()

olamundo()
```

A primeira função a ser chamada é a `olamundo()`, que por sua vez chama a função `ola()`, então a função `olamundo()` é empilhada, pois seu término depende do término da função `ola()`. Essa, por sua vez, chama a função `mundo()` e é empilhada. Ao terminar a execução da função `mundo()`, a função `ola()` é desempilhada e sua execução termina de onde havia parado (chamada à função `mundo()`). Como não há mais nada a ser executado nessa função, sua execução termina, e a função `olamundo()`é desempilhada e sua execução continua, encerrando assim o programa.

```
                 ola()
olamundo()  -->  olamundo()  --> olamundo() -->   Fim do programa.
```

Outra utilização de pilhas é a verificação do balanceamento de parênteses. Como saber se os seguintes parênteses estão balanceados, isto é, se para cada abre-parênteses, existe um fecha-parênteses correspondente?

```
((((((((((((((((((()()()))))))())))()))()))()))((()))))
```

Uma forma bem simples de resolver esse problema é ler a sequência de parênteses, um por um e, a cada abre-parênteses que encontrarmos, vamos empilhá-lo. Quando encontrarmos um fecha-parênteses, devemos desempilhar um abre-parênteses, pois encontramos um fecha-parênteses correspondente a ele. Assim, ao final da avaliação da sequência acima, se ela estiver balanceada corretamente, não restarão elementos na pilha. Vejamos um exemplo mais simples:

```
(())()

                (
Pilha       (   (   (       (

Leitura     (   (   )   )   (   )   FIM
```

Repare agora em uma sequência desbalanceada:

```
(())(

                (
Pilha       (   (   (       (   (

Leitura     (   (   )   )   (   FIM
```

Perceba que a leitura de parênteses da entrada terminou, mas a pilha não estava mais vazia.

Implementação

Agora eu lhe pergunto: o que muda da implementação da fila para a implementação da pilha? A diferença é pouca, mas é muito importante. Assim como para as filas, para a implementação de pilhas podemos utilizar uma lista como estrutura para armazenamento dos dados. Basta agora definir como será o funcionamento:

1. Iremos inserir e retirar elementos do início da lista?
2. Ou iremos inserir e retirar elementos do final da lista?

Sabendo que em Python a inserção e remoção de elementos no final de uma lista é menos custoso do que fazer as mesmas operações no início dela, vamos adotar a segunda opção. Então, complete o código abaixo e teste em seu computador.

```
1  class Pilha(object):
2      def __init__(self):
3          self.dados = []
4
5      def empilha(self, elemento):
6          ____________________________________
7
8      def desempilha(self):
9          ____________________________________
10
11 p = Pilha()
12 p.empilha(10)
13 p.empilha(20)
14 p.empilha(30)
15 p.empilha(40)
16 print p.desempilha(),
17 print p.desempilha(),
18 print p.desempilha(),
19 print p.desempilha(),
```

O programa acima, se implementado corretamente, deverá mostrar o seguinte resultado na tela:

```
40 30 20 10
```

Para adicionar um elemento no final de uma lista, basta usar o método `append()`. E para remover o último elemento da lista? `.pop(tamanho_da_lista-1)` é o que devemos fazer. E para obter o tamanho da lista, podemos usar a função `len()`. Então:

```
1  def desempilha(self):
2      return self.dados.pop(len(dados)-1)
```

Ou então, podemos usar `self.dados.pop(-1)`. O acesso ao índice `-1`é a forma pythônica de se referir ao último elemento de uma sequência.

```
1  def desempilha(self):
2      return self.dados.pop(-1)
```

Mais alguma operação?

Outra operação fundamental que deve estar disponível em uma Pilha, é um método que retorne um booleano indicando se a pilha está vazia. Um jeito bem simples seria usando a *builtin* `len()` sobre a nossa lista interna `self.dados` e verificando se ela retorna `0`como resultado.

```
1  def vazia(self):
2      return len(self.dados) == 0
```

Finalizando…

Tanto a fila quanto a pilha são estruturas importantes pra caramba e sua implementação deve fornecer um bom desempenho, visto que elas são amplamente usadas nos programas que compõem o nosso dia-a-dia. As implementações vistas aqui nesse post possuem fins didáticos apenas. Embora funcionem de forma adequada para serem utilizadas, elas não utilizam os recursos mais adequados para obter o melhor desempenho. Para usar na prática, existem soluções prontas e muito mais recomendadas, como por exemplo o módulo python queue (http://docs.python.org/library/queue.html), que pode ser usado tanto para implementação de Filas quanto para a implementação de Pilhas.

Códigos completos

Abaixo, seguem os códigos completos para a Fila e para a Pilha.

Fila

```python
class Fila(object):
    def __init__(self):
        self.dados = []

    def insere(self, elemento):
        self.dados.append(elemento)

    def retira(self):
        return self.dados.pop(0)

    def vazia(self):
        return len(self.dados) == 0
```

Pilha

```python
class Pilha(object):
    def __init__(self):
        self.dados = []

    def empilha(self, elemento):
        self.dados.append(elemento)

    def desempilha(self):
        if not self.vazia():
            return self.dados.pop(-1)

    def vazia(self):
        return len(self.dados) == 0
```

- filas (https://pythonhelp.wordpress.com/tag/filas/)
- pilhas (https://pythonhelp.wordpress.com/tag/pilhas/)
- python (https://pythonhelp.wordpress.com/tag/python/)
- queue (https://pythonhelp.wordpress.com/tag/queue/)
- stack (https://pythonhelp.wordpress.com/tag/stack/)

14 comentários sobre “Filas e Pilhas em Python”

Rafael Cuerda disse:
30 de setembro de 2013 às 12:04 pm
Parabéns pelo trabalho! Explicações muito didáticas, já li várias partes do blog.

Responder

stummjr disse:
30 de setembro de 2013 às 1:28 pm
Valeu!

2. Responder

Cesar Augusto disse:
21 de outubro de 2013 às 8:28 pm
Show de bola. Belo trabalho.

3. Responder

Tuiã disse:
19 de dezembro de 2013 às 3:11 pm
Sensacional! Gostei muito!

Responder

4.

Diogo disse:
13 de março de 2014 às 4:29 pm
Grande post!
Mas fiquei com uma dúvida!
Tem como aplicar a estrutura de fila em um par ordenado?
Seria algo como inserir/retirar dois valores de uma fila(X,Y).

- Responder

  stummjr disse:
  14 de março de 2014 às 4:29 pm
  Poder, pode. Mas acho que existem estruturas mais simples e práticas para fazer isso, como uma tupla, lista ou até mesmo um valor do tipo complex.

- Responder

  stummjr disse:
  14 de março de 2014 às 4:35 pm
  Opa. Acho que não havia entendido bem a sua dúvida. Você quer inserir pares ordenados como elementos na fila, certo?

  Pode inserir sim. Desde que você represente cada par ordenado por um objeto. Você poderia usar uma tupla de dois elementos:

  ```
  f = fila.Fila()
  f.insere( (2, 3) )
  f.insere( (8, 5) )
  ```

Responder

5.

Diogo disse:
15 de março de 2014 às 9:56 am
Uhmm….
Certo,…. mas será que dá para colocar um limite nessa fila?
Posso completá-la com valores até o limite,esvaziar e completar novamente?

O porque disso é; quero fazer uso da biblioteca matplotlib para construir um gráfico com valores que chegam a todo momento, porém para tenho que determinar os pontos a serem inseridos ([vai de tanto –> até tanto]), se não, teria que instruir o matplotlib a esperar o processo acabar e só então usar o conjunto de dados gerado!

Para isso queria construir uma fila que se enche -> esvazia –> enche novamente até todos os elementos terem sidos passados para o gráfico!

Estive lendo sobre estrutura de fila e pilha porém sou novo(em programação) e estou tendo dificuldade em aplicar esse tipo de estrutura em um par ordenado!

Obrigado!

Responder

stummjr disse:
15 de março de 2014 às 10:40 am
Você poderia modificar a classe Fila ou então usar a classe queue (http://docs.python.org/2/library/queue.html), na qual você pode definir o tamanho máximo dela.

6. Responder

Diogo disse:
15 de março de 2014 às 12:29 pm
Vou estudar sobre a classe queue!

Outra dúvida!
Não achei uma explicação (ou não soube como procurar); é possível reutilizar valores imprimidos no shell?
Por exemplo monto uma "fila" a qual recebe valores, e caso, eu imprima esses valores no shell python é possível fazer uma leitura desses valores diretamente do shell?

Estou pensando em uma maneira para diminuir o consumo de memória na execução do gráfico!

Responder

stummjr disse:
19 de março de 2014 às 11:24 am
Se entendi corretamente a sua dúvida, não conheço alguma forma de obter os valores que foram impressos a partir do console. Mas porque você não armazena referências aos elementos da fila?

7. Responder

Diogo disse:
19 de março de 2014 às 7:53 pm
Armazenar referencias?
Seria algo como ler o que foi armazenado no buffer do programa? Como?
A ideia de usar o shell como "display" para ver/capturar os dados seria uma forma de "fazer multi-tarefas" mais dinâmica e simples!
Uma vez realizada a ação e "enviada" a informação ao shell subsequentes processos poderiam ser disparados sem a necessidade de relacionar ninguém diretamente!

8. Responder

Richard Muller Rolhano Nunes disse:
23 de junho de 2016 às 3:39 am
Muito bom!! Bem explicado e de fácil compreensão.

9. Responder

Gabriel Gelli Checchinato disse:
28 de abril de 2020 às 2:13 pm
Excelente post! Organizado, bem explicado, intuitivo e fácil de entender!

Responder