pushable

Release 0.1.0

Stephen Leach

Aug 19, 2023

CONTENTS:

This is a Python package that provides a wrapper class Pushable that turns ordinary iterators into "peekable & pushable" iterators. Pushable iterators act like dynamically expanding queues, allowing you to peek ahead or push items back onto a queue that is only expanded as far as necessary.

Click here for the full API.

CHAPTER

ONE

BASIC USAGE

We can turn any iterable/iterator into a pushable iterator using the constructor.

```
from pushable import Pushable
count_up = Pushable( range( 0, 5 ) )
```

We can use it like an ordinary iterator:

```
print( next( count_up ) )
# Prints 0
```

Or we can look-ahead to see what is coming:

```
whats_up_next = count_up.peek()
print( whats_up_next )
# Print 1
print( next( count_up ) )
# Also prints 1 because peek does not remove the item from the internal queue.
```

We can even push back items onto it:

```
count_up.push("cat")
count_up.push("dog")
print( list( count_up ) )
# Prints 'dog', 'cat', 2, 3, 4
```

CHAPTER

TWO

EXAMPLES

From an iterator such as a file-object, which will iterate over the lines in a file, create a peekable/pushable iterator. This can be useful for example when we want to know if the iterator still has contents or want a sneak peek at what is coming.

```
from pushable import Pushable

def read_upto_two_blank_lines( filename ):
   with open( filename ) as file:
      plines = Pushable( file )
      # Pushable iterators can be used as booleans in the natural way.
      while plines:
            line = next( plines )
            # peekOr makes it safe to look ahead.
            if line == '\n' and plines.peekOr() == '\n':
               # Two blank lines encountered.
               break
            else:
               yield line
```

It is also useful to perform "macro-like" transformation.

```
from pushable import Pushable

def translate( text, translations ):
   ptokens = Pushable( text.split() )
   while ptokens:
      token = next(ptokens)
      if token in translations:
            ptokens.multiPush( *translations[token].split() )
      else:
            yield token

print( ' '.join( translate( 'My name is MYNAME', {'MYNAME':'Fred Bloggs'} ) ) )
# Prints: My name is Fred Bloggs
```

CHAPTER

THREE

MORE COMPLEX USES

In addition to peeking and popping items, which risks raising a *StopIteration* exception if there's nothing left on the internal queue, we can utilise *peekOr* and *popOr* to deliver a default value instead. The default value is passed as an optional parameter and falls back to None.

We can also peek and pop multiple values using *multiPeekOr* and *multiPopOr*, which return generators. These support skipping over values so that you can get the 2nd and 3rd value without getting the first e.g.

```
(second, third) = Pushable("pqr").multiPop(skip=1, count=2)
print( second, third )
# Prints: q r
```

Lastly, we can push multiple items with *multiPush*:

```
count_up.multiPush("cat", "dog", "rabbit")
print( list( count_up ) )
# Prints: ['cat', 'dog', 'rabbit']
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search