Function `throws_to_chars()`

This file is part of the bowling project (bowling.ipynb).

Petr Pošík

Department of Cybernetics, FEE CTU in Prague

EECS, BE5B33PRG: Programming Essentials, 2015

Specifications

Here we shall develop function `throws_to_chars()` with the following specs:

- Input: a legal sequence of throws, i.e. list of pins knocked down by a player in his/her individual throws.
- Output: list of at most 21 characters representing the individual throws in 'graphical' form. Strikes shall be denoted by `'X'`, spares by `'/'`, as usual. In case of strike, the function shall insert a `' '` in front of `'X'`.

Let's start with some test again.

In [1]:
```
from testing import *
set_default_test_verbosity(0)
```

Simple frames

In [2]:
```
def test_throws_to_chars():
    test_equal(throws_to_chars([]), [], 'No chars for no throws')
```

In [3]:
```
def throws_to_chars(throws):
    """Return a list of characters representing the throws of a bowling player"""
    return []
test_throws_to_chars()
```

```
.
```

OK, we are up and running. Let's define some real test cases.

In [4]:
```
def test_throws_to_chars():
    test_equal(throws_to_chars([]), [], 'No chars for no throws')
    test_equal(throws_to_chars([1]), ['1'], 'Single simple throw')
    test_equal(throws_to_chars([1,4]), ['1', '4'], 'Two simple throws')
    test_equal(throws_to_chars([1,4,2]), ['1', '4', '2'], 'Three simple throws')
```

In [5]:
```
def throws_to_chars(throws):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    for throw in throws:
        chars.append(str(throw))
    return chars
test_throws_to_chars()
```

```
....
```

Great, simple cases are not hard. What about some strikes?

Strikes

In [6]:
```
def test_throws_to_chars():
    test_equal(throws_to_chars([]), [], 'No chars for no throws')
    test_equal(throws_to_chars([1]), ['1'], 'Single simple throw')
    test_equal(throws_to_chars([1,4]), ['1', '4'], 'Two simple throws')
    test_equal(throws_to_chars([1,4,2]), ['1', '4', '2'], 'Three simple throws')
    # Strikes
    test_equal(throws_to_chars([10]), [' ', 'X'], 'Single strike')
    test_equal(throws_to_chars([10,10]), [' ', 'X', ' ', 'X'], 'Pair of strikes')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,10,10,10]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Perfect game')
test_throws_to_chars()
```

```
....
Test 'Single strike' at line 7 FAILED.
  [' ', 'X'] expected, but got ['10'].

Test 'Pair of strikes' at line 8 FAILED.
  [' ', 'X', ' ', 'X'] expected, but got ['10', '10'].

Test 'Perfect game' at line 11 FAILED.
  [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
', 'X', 'X', 'X', 'X'] expected, but got ['10', '10', '10', '10', '10', '10', '10', '
10', '10', '10', '10', '10'].
```

Obviously, we are not able to handle strikes. Let's try it!

In [7]:
```
def throws_to_chars(throws):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    for throw in throws:
        if throw == 10:       # Strike!
            chars.append(' ')
            chars.append('X')
        else:
            chars.append(str(throw))
    return chars
test_throws_to_chars()
```

```
......
Test 'Perfect game' at line 11 FAILED.
  [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
', 'X', 'X', 'X', 'X'] expected, but got [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' '
, 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X'].
```

That's better but we are not able to handle the spacial case of the last frame. We must somehow track the frame number and use a special handling of the last 10th frame (or, in other words, we should add the space only in frames 1-9).

In [8]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    for throw in throws:
        if throw == 10:        # Strike!
            if frame < n_frames:
                chars.append(' ')
            chars.append('X')
            frame += 1
        else:
            chars.append(str(throw))
            frame += 1
    return chars
test_throws_to_chars()
```

```
.......
```

The tests pass, but the code still may not work well for combinations of simple frames and strikes. Let's add some tests combining these two.

In [9]:
```
def test_throws_to_chars(verbosity=1):
    test_equal(throws_to_chars([]), [], 'No chars for no throws')
    test_equal(throws_to_chars([1]), ['1'], 'Single simple throw')
    test_equal(throws_to_chars([1,4]), ['1', '4'], 'Two simple throws')
    test_equal(throws_to_chars([1,4,2]), ['1', '4', '2'], 'Three simple throws')
    # Strikes
    test_equal(throws_to_chars([10]), [' ', 'X'], 'Single strike')
    test_equal(throws_to_chars([10,10]), [' ', 'X', ' ', 'X'], 'Pair of strikes')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,10,10,10]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Perfect game')
    test_equal(throws_to_chars([1,2,10]), ['1', '2', ' ', 'X'], 'Strike after simple fr
ame')
    test_equal(throws_to_chars([10,1,2]), [' ', 'X', '1', '2'], 'Simple frame after str
ike')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,1,2]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', '1', '2'],
               'Perfect game spoiled at the end')
    test_equal(throws_to_chars([1,2,10,10,10,10,10,10,10,10,10,10,10,10]),
               ['1', '2', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Bad first frame followed by a perfect game')
    test_equal(throws_to_chars([1,2,10,3,4,10,10,10,10,10,10,10,10,5]),
               ['1', '2', ' ', 'X', '3', '4', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', '5'],
               'Mixed simple frames and strikes')
test_throws_to_chars()
```

```
.........
Test 'Bad first frame followed by a perfect game' at line 19 FAILED.
  ['1', '2', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
', 'X', 'X', 'X', 'X'] expected, but got ['1', '2', ' ', 'X', ' ', 'X', ' ', 'X', ' '
, 'X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X'].

Test 'Mixed simple frames and strikes' at line 22 FAILED.
  ['1', '2', ' ', 'X', '3', '4', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
', 'X', 'X', 'X', '5'] expected, but got ['1', '2', ' ', 'X', '3', '4', ' ', 'X', ' '
, 'X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', '5'].
```

Well, the code increments the frame counter after a each single simple throw, i.e. 2 times in a single simple frame. The frame counter shall be incremented only when a frame is finished. Let's introduce another variable there.

In [10]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    for throw in throws:
        if throw == 10:
            if frame < n_frames:
                chars.append(' ')
            chars.append('X')
            frame += 1
        else:
            chars.append(str(throw))
            if not first_ball_in_frame:
                frame += 1
            first_ball_in_frame = not first_ball_in_frame
    return chars
test_throws_to_chars()
```

```
............
```

Great! All tests pass. However, the code starts to look messy. But bear with it a while, we will clean it up in the end.

Spares

We are still not able to handle any spares. Let's add a first simple case:

In [11]:
```
def test_throws_to_chars():
    test_equal(throws_to_chars([]), [], 'No chars for no throws')
    test_equal(throws_to_chars([1]), ['1'], 'Single simple throw')
    test_equal(throws_to_chars([1,4]), ['1', '4'], 'Two simple throws')
    test_equal(throws_to_chars([1,4,2]), ['1', '4', '2'], 'Three simple throws')
    # Strikes
    test_equal(throws_to_chars([10]), [' ', 'X'], 'Single strike')
    test_equal(throws_to_chars([10,10]), [' ', 'X', ' ', 'X'], 'Pair of strikes')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,10,10,10]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Perfect game')
    test_equal(throws_to_chars([1,2,10]), ['1', '2', ' ', 'X'], 'Strike after simple fr
ame')
    test_equal(throws_to_chars([10,1,2]), [' ', 'X', '1', '2'], 'Simple frame after str
ike')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,1,2]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', '1', '2'],
               'Perfect game spoiled at the end')
    test_equal(throws_to_chars([1,2,10,10,10,10,10,10,10,10,10,10,10,10]),
               ['1', '2', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Bad first frame followed by a perfect game')
    test_equal(throws_to_chars([1,2,10,3,4,10,10,10,10,10,10,10,10,5]),
               ['1', '2', ' ', 'X', '3', '4', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', '5'],
               'Mixed simple frames and strikes')
    # Spares
    test_equal(throws_to_chars([1,9]), ['1', '/'], 'Single spare')
test_throws_to_chars()
```

```
............
Test 'Single spare' at line 24 FAILED.
  ['1', '/'] expected, but got ['1', '9'].
```

What can we do to pass the test? It seems that the output is almost OK, but we got `9` instead of `/`. Seems that we have to check whether the sum of last 2 throws is 10, and in that case put `/` in the chars instead the second throw. For that purpose, however, we need to remember the most recent throw. Let's introduce a new variable `last_throw`.

In [12]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    last_throw = 0
    for throw in throws:
        if throw == 10:      # Strike!
            if frame < n_frames:
                chars.append(' ')
            chars.append('X')
            frame += 1
        else:
            if throw + last_throw == 10:   # Spare!
                chars.append('/')
            else:
                chars.append(str(throw))
            last_throw = throw
            if not first_ball_in_frame:
                frame += 1
            first_ball_in_frame = not first_ball_in_frame
    return chars
test_throws_to_chars()
```

```
.............
```

Great! The tests pass now, but I think that there is still a mistake. This shall show up when the sum of the second throw of the (i-1)-th frame and the first throw of the i-th frame will be equal to 10. Let's create a test case for this:

In [13]:
```
def test_throws_to_chars():
    test_equal(throws_to_chars([]), [], 'No chars for no throws')
    test_equal(throws_to_chars([1]), ['1'], 'Single simple throw')
    test_equal(throws_to_chars([1,4]), ['1', '4'], 'Two simple throws')
    test_equal(throws_to_chars([1,4,2]), ['1', '4', '2'], 'Three simple throws')
    # Strikes
    test_equal(throws_to_chars([10]), [' ', 'X'], 'Single strike')
    test_equal(throws_to_chars([10,10]), [' ', 'X', ' ', 'X'], 'Pair of strikes')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,10,10,10]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Perfect game')
    test_equal(throws_to_chars([1,2,10]), ['1', '2', ' ', 'X'], 'Strike after simple fr
ame')
    test_equal(throws_to_chars([10,1,2]), [' ', 'X', '1', '2'], 'Simple frame after str
ike')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,1,2]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', '1', '2'],
               'Perfect game spoiled at the end')
    test_equal(throws_to_chars([1,2,10,10,10,10,10,10,10,10,10,10,10]),
               ['1', '2', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Bad first frame followed by a perfect game')
    test_equal(throws_to_chars([1,2,10,3,4,10,10,10,10,10,10,10,10,5]),
               ['1', '2', ' ', 'X', '3', '4', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', '5'],
               'Mixed simple frames and strikes')
    # Spares
    test_equal(throws_to_chars([1,9]), ['1', '/'], 'Single spare')
    test_equal(throws_to_chars([1,9,1]), ['1', '/', '1'], 'Spare + sum 10 across frames
')
    test_equal(throws_to_chars([2,7,3]), ['2', '7', '3'], 'Non-spare + sum 10 across fr
ames')
    test_equal(throws_to_chars([0,10,0]), ['0', '/', '0'], 'Zero-Spare-Zero')
    test_equal(throws_to_chars([10,0,10]), [' ', 'X', '0', '/'], 'Strike-Zero-Spare')
test_throws_to_chars()
```

```
............
Test 'Spare + sum 10 across frames' at line 25 FAILED.
  ['1', '/', '1'] expected, but got ['1', '/', '/'].

Test 'Non-spare + sum 10 across frames' at line 26 FAILED.
  ['2', '7', '3'] expected, but got ['2', '7', '/'].

Test 'Zero-Spare-Zero' at line 27 FAILED.
  ['0', '/', '0'] expected, but got ['0', ' ', 'X', '0'].

Test 'Strike-Zero-Spare' at line 28 FAILED.
  [' ', 'X', '0', '/'] expected, but got [' ', 'X', '0', ' ', 'X'].
```

As expected, these new tests failed. The condition checking for spare does not take into account the frame boundaries. We can take advantage of the `first_ball_in_frame` variable, which is set to `False` on the second throw in a frame.

In [14]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    last_throw = 0
    for throw in throws:
        if throw == 10:      # Strike !
            if frame <= 9:
                chars.append(' ')
            chars.append('X')
            frame += 1
        else:
            if not first_ball_in_frame and throw + last_throw == 10:   # Spare!
                chars.append('/')
            else:
                chars.append(str(throw))
            last_throw = throw
            if not first_ball_in_frame:
                frame += 1
            first_ball_in_frame = not first_ball_in_frame
    return chars
test_throws_to_chars()
```

```
..............
Test 'Zero-Spare-Zero' at line 27 FAILED.
  ['0', '/', '0'] expected, but got ['0', ' ', 'X', '0'].

Test 'Strike-Zero-Spare' at line 28 FAILED.
  [' ', 'X', '0', '/'] expected, but got [' ', 'X', '0', ' ', 'X'].
```

Well, that helped, but not completely. The test for strike must also be augmented to fire only when a 10 is thrown on the first ball of frame.

In [15]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    last_throw = 0
    for throw in throws:
        if first_ball_in_frame and throw == 10:   # Strike !
            if frame <= 9:
                chars.append(' ')
            chars.append('X')
            frame += 1
        else:
            if not first_ball_in_frame and throw + last_throw == 10:  # Spare !
                chars.append('/')
            else:
                chars.append(str(throw))
            last_throw = throw
            if not first_ball_in_frame:
                frame += 1
            first_ball_in_frame = not first_ball_in_frame
    return chars
test_throws_to_chars()
```

```
..............
```

Let's introduce a few other tests to check the function:

In [16]:
```
def test_throws_to_chars():
    test_equal(throws_to_chars([]), [], 'No chars for no throws')
    test_equal(throws_to_chars([1]), ['1'], 'Single simple throw')
    test_equal(throws_to_chars([1,4]), ['1', '4'], 'Two simple throws')
    test_equal(throws_to_chars([1,4,2]), ['1', '4', '2'], 'Three simple throws')
    # Strikes
    test_equal(throws_to_chars([10]), [' ', 'X'], 'Single strike')
    test_equal(throws_to_chars([10,10]), [' ', 'X', ' ', 'X'], 'Pair of strikes')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,10,10,10]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Perfect game')
    test_equal(throws_to_chars([1,2,10]), ['1', '2', ' ', 'X'], 'Strike after simple fr
ame')
    test_equal(throws_to_chars([10,1,2]), [' ', 'X', '1', '2'], 'Simple frame after str
ike')
    test_equal(throws_to_chars([10,10,10,10,10,10,10,10,10,1,2]),
               [' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', '1', '2'],
               'Perfect game spoiled at the end')
    test_equal(throws_to_chars([1,2,10,10,10,10,10,10,10,10,10,10,10,10]),
               ['1', '2', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', 'X'],
               'Bad first frame followed by a perfect game')
    test_equal(throws_to_chars([1,2,10,3,4,10,10,10,10,10,10,10,10,5]),
               ['1', '2', ' ', 'X', '3', '4', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X', '
 ', 'X', ' ', 'X', 'X', 'X', '5'],
               'Mixed simple frames and strikes')
    # Spares
    test_equal(throws_to_chars([1,9]), ['1', '/'], 'Single spare')
    test_equal(throws_to_chars([1,9,1]), ['1', '/', '1'], 'Spare + sum 10 across frames
')
    test_equal(throws_to_chars([2,7,3]), ['2', '7', '3'], 'Non-spare + sum 10 across fr
ames')
    test_equal(throws_to_chars([0,10,0]), ['0', '/', '0'], 'Zero-Spare-Zero')
    test_equal(throws_to_chars([10,0,10]), [' ', 'X', '0', '/'], 'Strike-Zero-Spare')
    # Additional tests
    test_equal(throws_to_chars([1,9,2,8,3,7,4,6,5,5,6,4,7,3,8,2,9,1,10,10,10]),
               ['1','/','2','/','3','/','4','/','5','/','6','/','7','/','8','/','9','/'
,'X','X','X'],
               'Spares and strikes')
    test_equal(throws_to_chars([1,4,0,10,6,4,5,5,10,0,1,7,3,7,3,10,2,8,6]),
               ['1','4','0','/','6','/','5','/',' ','X','0','1','7','/','7','/',' ','X'
,'2','/','6'],
               'Normal game 1')
    test_equal(throws_to_chars([1,4,4,5,6,4,5,5,10,0,1,7,3,6,4,10,2,8,6]),
               ['1','4','4','5','6','/','5','/',' ','X','0','1','7','/','6','/',' ','X'
,'2','/','6'],
               'Normal game 2')
test_throws_to_chars()
```

```
..................
```

Hmmm, it seems that our function works! But its code is not really readable. Let's refactor it a bit.

Refactoring

The purpose of refactoring is to make the code better, more readable, more intention-revealing, without changing its function. We have tests, so we can afford to make changes to the code.

The first `if` in the `for` loop tests whether a strike happened. The body of the `if` then handles the strike. Let's separate these two things into short nested functions.

In [17]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    last_throw = 0

    def strike():
        return first_ball_in_frame and throw == 10

    def process_strike():
        nonlocal frame, chars
        if frame < n_frames:
            chars.append(' ')
        chars.append('X')
        frame += 1

    for throw in throws:
        if strike():
            process_strike()
        else:
            if not first_ball_in_frame and throw + last_throw == 10:
                chars.append('/')
            else:
                chars.append(str(throw))
            last_throw = throw
            if not first_ball_in_frame:
                frame += 1
            first_ball_in_frame = not first_ball_in_frame
    return chars
test_throws_to_chars()
```

```
..................
```

The code works the same way as in previous case, but by extracting 2 parts of code and giving them meaningful names, the main body of the function is much more readable.

Let's continue with this process and let's separate also the `else`-block into a function, and the part of code that increments frame as well:

In [18]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    last_throw = 0

    def strike():
        return first_ball_in_frame and throw == 10

    def process_strike():
        nonlocal frame, chars
        if frame < n_frames:
            chars.append(' ')
        chars.append('X')
        frame += 1

    def spare():
        return not first_ball_in_frame and throw + last_throw == 10

    def update_frame():
        nonlocal last_throw, first_ball_in_frame, frame
        last_throw = throw
        if not first_ball_in_frame:
            frame += 1
        first_ball_in_frame = not first_ball_in_frame

    for throw in throws:
        if strike():
            process_strike()
        else:
            if spare():
                chars.append('/')
            else:
                chars.append(str(throw))
            update_frame()
    return chars
test_throws_to_chars()
```

```
..................
```

This is already quite readable, and we may be happy with that. Somebody may consider the following version even more clear and intention revealing:

In [19]:
```
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    last_throw = 0

    def strike():
        return first_ball_in_frame and throw == 10

    def process_strike():
        nonlocal frame, chars
        if frame < n_frames:
            chars.append(' ')
        chars.append('X')
        frame += 1

    def spare():
        return not first_ball_in_frame and throw + last_throw == 10

    def update_frame():
        nonlocal last_throw, first_ball_in_frame, frame
        last_throw = throw
        if not first_ball_in_frame:
            frame += 1
        first_ball_in_frame = not first_ball_in_frame

    def process_spare():
        nonlocal chars
        chars.append('/')
        update_frame()

    def process_simple_throw():
        nonlocal chars
        chars.append(str(throw))
        update_frame()


    for throw in throws:
        if strike():
            process_strike()
        elif spare():
            process_spare()
        else:
            process_simple_throw()
    return chars
test_throws_to_chars()
```

```
..................
```

With all these refactorings, the code gets lengthy, but also more modular, and easier to understand. The last version of the main function has a crystal clear structure on the top level!

All these refactorings are actually made possible thanks to the fact we have a test suite which would detect any (well, not exactly true) error we could make during the refactorings.

Output module just for reuse

In [20]:
```
%%writefile bowling_1.py
def throws_to_chars(throws, n_frames=10):
    """Return a list of characters representing the throws of a bowling player"""
    chars = []
    frame = 1     # We enter game in the first frame
    first_ball_in_frame = True
    last_throw = 0

    def strike():
        return first_ball_in_frame and throw == 10

    def process_strike():
        nonlocal frame, chars
        if frame < n_frames:
            chars.append(' ')
        chars.append('X')
        frame += 1

    def spare():
        return not first_ball_in_frame and throw + last_throw == 10

    def update_frame():
        nonlocal last_throw, first_ball_in_frame, frame
        last_throw = throw
        if not first_ball_in_frame:
            frame += 1
        first_ball_in_frame = not first_ball_in_frame

    def process_spare():
        nonlocal chars
        chars.append('/')
        update_frame()

    def process_simple_throw():
        nonlocal chars
        chars.append(str(throw))
        update_frame()


    for throw in throws:
        if strike():
            process_strike()
        elif spare():
            process_spare()
        else:
            process_simple_throw()
    return chars
```

```
Overwriting bowling_1.py
```

Notebook config

Some setup follows. Ignore it.

In [21]:
```
from notebook.services.config import ConfigManager
cm = ConfigManager()
cm.update('livereveal', {
              'theme': 'Simple',
              'transition': 'slide',
              'start_slideshow_at': 'selected',
              'width': 1268,
              'height': 768,
              'minScale': 1.0
})
```

Out[21]:
```
{'height': 768,
 'minScale': 1.0,
 'start_slideshow_at': 'selected',
 'theme': 'Simple',
 'transition': 'slide',
 'width': 1268}
```

In [22]:
```
%%HTML
<style>
.reveal #notebook-container { width: 90% !important; }
.CodeMirror { max-width: 100% !important; }
pre, code, .CodeMirror-code, .reveal pre, .reveal code {
    font-family: "Consolas", "Source Code Pro", "Courier New", Courier, monospace;
}
pre, code, .CodeMirror-code {
    font-size: inherit !important;
}
.reveal .code_cell {
    font-size: 130% !important;
    line-height: 130% !important;
}
</style>
```

In [ ]: