The Accumulator Pattern

Loops are used to examine sequences such as strings and accumulate information about the items in the sequence. For example, we might want to count the number of vowels in a string, or build a new string containing only the punctuation in a string, or examine a string to find out whether it consists only of digits.

The accumulator pattern describes the approach. We'll use the accumulator pattern a lot over the rest of the course.

Accumulator pattern

You will use an "accumulator variable" to build up the answer. At the beginning, you haven't accumulated anything, so:

- If you're counting something, start the accumulator at `0`.
- If you're building a new string, start the accumulator at `''`.
- If you're checking for a property of the characters in the string, start the accumulator at `True` or `False`, depending on the question.

And the steps:

1. Initialize the accumulator variable.
2. Write a loop that examines each character in the string in turn.
    - If the current character is interesting, update the accumulator.
3. Return the accumulator.

Three related examples

[Learning tip: try using the debugger to watch the control flow.]

```
def count_digits(s: str) -> int:
    """ Return the number of characters in s that are digits.

    [Note: if a character is a digit, it's interesting. If it isn't a digit,
    it isn't interesting for this problem. What is interesting depends on the
    problem you're trying to solve.]

    >>> count_digits('12ab')
    2
    >>> count_digits('#$*^W')
    0
    >>> count_digits('5t3&*1')
    3
    """

    count = 0                  # Initialize the accumulator

    for char in s:             # Examine each character in s, one at a time
```

```
        if char.isdigit():     #     If the character is interesting
            count = count + 1  #         Update the accumulator

    return count               # When we reach here, the for loop has
                               # terminated, which means we've examined every
                               # character in s.
```

[Learning tip: many people want to have an else part. With the accumulator pattern, we often don't have an else because we only want to do something when we find an interesting character.]

We can use the exact same structure to build up a string of digits rather than just counting them. We'll use a string accumulator. The rest of the machinery is identical!

```
def get_digits(s: str) -> int:
    """ Return a new string containing the characters in s that are digits,
    in the order they appear in s.

    >>> get_digits('12ab')
    '12'
    >>> get_digits('#$*^W')
    ''
    >>> get_digits('5t3&*1')
    '531'
    """

    digits = ''                    # Initialize the accumulator

    for char in s:                 # Examine each character in s, one at a time
        if char.isdigit():         #     If the character is interesting
            digits = digits + char #         Update the accumulator

    return digits
```

We can use the exact same structure to find out whether any characters are digits. We'll use a Boolean accumulator.

```
def has_digits(s: str) -> int:
    """ Return True iff at least one of the characters in s is a digit.

    >>> has_digits('12ab')
    True
    >>> has_digits('#$*^W')
    False
    >>> has_digits('5t3&*1')
    True
    """

    digits = False            # Initialize the accumulator. At the
                              # beginning, we haven't seen any digits.
```

```
    for char in s:            # Examine each character in s, one at a time
        if char.isdigit():    #     If the character is interesting
            digits = True     #         Update the accumulator

    return digits
```

[Learning tip: with Booleans, many people want to use return to exit early (which can work), or have an else part that does something, like return False (which doesn't work). Try both and step through them in the debugger.]

While loops

While loops can be used to loop over the characters in a string, too. The accumulator logic is identical, but the machinery to get the characters one at a time requires us to loop over the indexes of the characters in the string rather than looping over the characters themselves.

[Learning tip: identify exactly which parts are the accumulator pattern and what is just the machinery you need to iterate over the characters in a string using a while loop.]

```
def count_digits(s: str) -> int:
    """ Return the number of characters in s that are digits.
    >>> count_digits('12ab')
    2
    >>> count_digits('#$*^W')
    0
    >>> count_digits('5t3&*1')
    3
    """

    count = 0                 # Initialize the accumulator

    i = 0                     # The index in s to examine next.
    while i != len(s):        # We are done when i == len(s), so we're not
                              # done while it's not equal.
        char = s[i]           # Get the next character. The if statement
                              # below is identical.
        if char.isdigit():    #     If the character is interesting
            count = count + 1 #         Update the accumulator

        i = i + 1             # We need to move on to the next index.

    return count              # When we reach here, the while loop has
                              # terminated, which means we've examined every
                              # character in s.
```