□for loops

Tuesday, February 15, 2022 3:12 PM

Use for loops when you know when the loop ends (working with a string, a limited list, etc.

```
for item in iterable :        ← Colon
    do something              ← Indent
```

where : do something = block of code we want to execute
item = the name (new) of one or more variables
The variable will be bound to each of the items in the sequence in turn.
iterable = specify what the values are in
* An iterable is an object that can be iterated over.
e.g: Strings, Lists.

Examples : Count the number of occurences in a string
chrome_4 = "ATG6AACT..."

```
while loop :
i = 0
count = 0
while i < len(chrome_4):
    if chrome_4[i] == 'A':
        count += 1
    i += 1
```

```
for loop
count = 0
for character in chrome_4:
    if character == 'A':
        count += 1
```

⇒ Difference : We needed (i) to define number of loops as well as the index of each character

Write a function that takes in a string and returns the number of vowels in the string.

```
def count_vowels(s):
    """
    (str) -> int
    Return the number of vowles in s.
    """
    num_vowels = 0

    for letter in s:
        if letter in 'aieou':
            num_vowels += 1

    return num_vowels
```

Write a function to return the unique non-numeric characters in a string.

```
def find_chars(s):
    """
    (str) -> str
    Return a concatentation of all unique seperators in s.
    """
    seperators = ""
    for character in s:
        if character.isnumeric() == False:
            seperators += character
    '''

    return seperators
```

```
print(find_chars('23,613-23;2:45'))
```

.::

Loop on indices

Friday, 18 February 2022 13:33

We know that using while loop, we can loop over indices of a string.

eg
```
while i < len(x):
    print(i, x[i])
    i += 1
```

But when using for loop, we iterate over the values, not the indices

⇒ Can we use for loop to loop over indices?

• Looping on a range()

+) Python has built-in function called **range()** that can be used to generate **a sequence of numbers**. The general syntax of range is as follows

```
range(start, stop, step)
```

+) Similar to the string slicing syntax:
- **Stop value is not included** in the sequence of numbers generated
- Can omit start and step which will result in default values being used `range(n)` → `range(0, n, 1)`

+) `range()` is typically used in a for loop to **iterate over a sequence of numbers**.

+) `range()` is iterable

⇒ Now we have a sequence of numbers that matches the indices of a string

In [10]:
```
chrome_4 = "ATGGGCAATCGATGGCCTAATCTCTCTAAG"

for i in range(1, len(chrome_4)):
    print(i, chrome_4[i])
```

```
1 T
2 G
3 G
4 G
5 C
6 A
7 A
8 T
9 C
10 G
11 A
12 T
13 G
14 G
15 C
16 C
17 T
18 A
19 A
20 T
21 C
22 T
23 C
24 T
25 C
26 T
27 A
28 A
29 G
```

In [14]:
```
for i in range(len(chrome_4) // 2):
    print(chrome_4[i], end=" ")
```

```
A T G G G C A A T C G A T G G
```

Example 1 Add up all even numbers between 1 & 100 using for loop

```
>>> total = 0
    for i in range(101):
        if i % 2 == 0:
```

⇒ 2550

```
total += i
```

Example 2 count adjacent repeats (how many adjacent pairs are there)

```
def count_adjacent_repeats(s):
    for num in range(len(s) + 1):
        if s[num] == s[num + 1]:
            count += 1
    return count

print(count_adjacent_repeats("abbefggha"))
```

→ 2 ✓

```
def count_adjacent_repeats(s):
    for num in range(len(s)):
        if s[num] == s[num - 1]:
            count += 1
    return count

print(count_adjacent_repeats("abbefggha"))
```

→ 3 XX

Why ?? Because s[0] == s[-1]