Ch05-Iterations

September 10, 2025

1 Iterations

- http://openbookproject.net/thinkcs/python/english3e/iteration.html

1.1 Topics

- what is iteration/loop and why do we need it?
- types of loops and their syntaxes
- various keywords used with loops
- quick applications

1.2 Iterations / Loops

- iteratations are also called loops
- life is full of loops; everyday routines such as going to college, cooking food, slicing vegetables, etc.
- loops make computer repeatedly execute some block of code
- working with loop has prep work to start it and some condition to end it
- two types of loops: **for** and **while**
  - some keywords that may appear in loops:
  - **for, while, break, continue, in, for … else, while… else**

1.2.1 Slice a carrot

1. get a cutting board
2. get a knife
3. get a carrot
4. place the carrot on cutting board
5. lift knife
6. advance carrot
7. slice carrot
8. if you see only your finger, then quit
9. else go to step 4

- steps 4-8 form a loop!
- steps 1-3 preparation
- step 8 checks condition to end the loop
- steps 4-7 are tasks that need to be iterated

1.3 for loop

- works with a range of values
- syntax

```
for val in rangeofValues:
    # loop body
```

- useful when you know exactly how many times the loop should be executed
- commonly used with built-in **range( )** function
- typically, variables **i**, **j**, etc. are used as loop counters

[1]:
```
# let's learn about range()
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, …, j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __bool__(self, /)
 |      True if self else False
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __reduce__(…)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(…)
 |      Return a reverse iterator.
 |
 |  count(…)
 |      rangeobject.count(value) -> integer -- return number of occurrences of
value
 |
 |  index(…)
 |      rangeobject.index(value) -> integer -- return index of value.
 |      Raise ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  start
 |
 |  step
 |
 |  stop
```

[2]:
```
# convert range into list of values
# learn about list in later chapter
list(range(10))
```

[2]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[3]:
```
list(range(1, 11))
```

[3]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[4]:
```
# let's provide a step of 2
list(range(1, 11, 2))
```

[4]:
```
[1, 3, 5, 7, 9]
```

[5]:
```
# print hello world some number of times!
for i in range(1, 11):
    print(i, 'hello world')
```

```
1 hello world
2 hello world
3 hello world
4 hello world
5 hello world
6 hello world
7 hello world
8 hello world
9 hello world
10 hello world
```

1.3.1 Visualize with PythonTutor.com

[6]:
```
for num in range(20):
    print(num)


print('done')
```

```
0
1
2
3
4
5
6
7
```

```
8
9
10
11
12
13
14
15
16
17
18
19
done
```

1.3.2 continue loop

- **continue** statement skips the rest of the current loop when it is executed

[7]:
```
# for loop example - continue
for i in range(1, 11): # range ... 1...10
    if i%2 == 0:
        continue
    print(i)
```

```
1
3
5
7
9
```

1.3.3 Visualize with PythonTutor.com

1.3.4 break loop

- **break** statement is used to break a loop
- it breaks or exits the loop immidiately
- rest of the current loop and the rest of the loop will not be executed!

[8]:
```
# for loop example - break
for i in range(10):
    if i == 10:
        break
    print(i)

print('done')
```

```
0
1
2
3
```

```
4
5
6
7
8
9
done
```

1.4 for … else

- you can write an optional else statement with for loop
- the else clause executes when the loop completes normally
    - this means that the loop did not encounter any **break** statement

[9]:
```
for i in range(5):
    if i == 2:
        continue
    print(i)
else:
    print('end!')
```

```
0
1
3
4
end!
```

[10]:
```
for i in range(5):
    if i == 2:
        break
    print(i)
else:
    print('end!')
```

```
0
1
```

[11]:
```
# for... else example
# write a program to test whether a given number is prime
n = 97
#isPrime = True
for i in range(2, n//2+1):
    if n%i == 0:
        #isPrime = False
        print(n, ' is not prime')
        break
else:
    print(n, 'is prime')
```

```
97 is prime
```

1.5 while loop

- while loop is used when you're not sure the exact number of times the loop should be executed
  - loop may execute 0 or many times based on the contidion!
- syntax:

```
while <condition>:
    # loop body
```

[12]:
```python
# infinite loop - will never stop; uncomment and run to see for yourself!
# Interrupt Kernel on Jupyter Notebook; enter Ctrl+C on Terminal to stop␣
↪infinite loop
#while True:
#    print('hello')
```

[13]:
```python
i = 1
while i <= 5:
    print(i, 'Hello World')
    i += 1
```

```
1 Hello World
2 Hello World
3 Hello World
4 Hello World
5 Hello World
```

1.5.1 Visualize with PythonTutor.com

1.5.2 While loop applications

- loops have many applications in coding
  - it's a foundational block!

input validation

- data types and values need to be often validated for correct results
- while loop can be used for input validation

[14]:
```python
# not sure when the user will get this correct...
# at least 1 try; could be more!!
while True:
    number = input('Enter a whole number: ')
    if not number.isdigit():
        print('Not a valid number!')
        continue
    number = int(number)
    break
```

```
Enter a whole number:
Not a valid number!
Enter a whole number: 23
```

[15]:
```
print(f'You entered {number}')
```

```
You entered 23
```

countdown

- run countdown as a script
- uses `time.sleep(sec)` to sleep for a second
- `os.system('clear')` clears the console
- `os.name` stores the type of operating system
  - types of os are: `'posix'`, `'nt'`, `'mac'`, `'os2'`, `'ce'`, `'java'`, `'riscos'`

[16]:
```
# while loop - countdown
import time
import os

def clearScreen():
    if os.name == 'nt':
        os.system('cls')
    else:
        os.system('clear')

i = 10
while i >= 1:
    print(i)
    time.sleep(1) # sleep for 1 second
    clearScreen()
    i -= 1

#time.sleep(1)
print('Hapy new year!')
```

```
10
9
8
7
6
5
4
3
2
1
Hapy new year!
```

1.6 while... else

- optional **else** clause executes when the loop executes normally
  - meaning the loop didn't encounter any **break** statement

[17]:
```python
while True:
    print('Hello')
    break
else:
    print("Look loop did encounter break")
```

```
Hello
```

[18]:
```python
count = 1
while count < 5:
    print(count)
    count += 1
else:
    print("Look loop didn't encounter break")
```

```
1
2
3
4
Look loop didn't encounter break
```

1.7 Exercises

Exercise 1. Write a program to count the number of digits in a positive integer

[19]:
```python
n = 1234
print(len(str(n)))
```

```
4
```

Exercise 1.1. Convert Exercise 1 to a function and write at least two test cases.

[20]:
```python
def countDigits(number):
    """
    given a positive number the function return number of digits
    in the number
    """
    pass
```

[21]:
```python
# test cases
def test_countDigits():
    assert countDigits(10) == 2
    assert countDigits(19923) == 5
    assert countDigits(34324324327890) == 14
    print('all test cases passed!')
```

Exercise 2. Write a program to count the number of digits 0 or 5 in a positive ingeger.

[22]:
```python
# solution 1
# converst positive integer into string and parse one character at a time
```

```
n = 12345505
n = str(n)
count0 = 0
count5 = 0
for c in n:
    if c == '0':
        count0 += 1
    elif c == '5':
        count5 += 1
print('There are {} zero(s) and {} five(s) digits in {}.'.format(count0,␣
 ↪count5, n))
```

```
There are 1 zero(s) and 3 five(s) digits in 12345505.
```

[23]:
```
# solution 2
# repeated divide the positive integer by 10 counting the value of remainder␣
 ↪until the number is greater than 0
n = 12345505
savedN = n
count0 = 0
count5 = 0
while n > 0:
    n, rem = divmod(n, 10)
    if rem == 0:
        count0 += 1
    elif rem == 5:
        count5 += 1

print('There are {} zero(s) and {} five(s) digits in {}.'.format(count0,␣
 ↪count5, savedN))
```

```
There are 1 zero(s) and 3 five(s) digits in 12345505.
```

Exercise 2.1. Convert Exercise 2 to a function and write at least 2 test cases

Exercise 3. FizzBuzz program - Write a program that prints numbers between 1 and 100 with the following requirements. If the number is divisible by 3, print 'Fizz'. If the number is divisible by 5, print 'Buzz'. If the number is divisible by both 3 and 5, print 'FizzBuzz'.

Exercise 3.1. Convert Exercise 3 into a function and write at least 2 three test cases.

Exercise 4. What are the output(s) of the following code snippets?

[24]:
```
a = 11
while a < 10:
    print(a)
    a += 1
    #break
else:
    print('Done')
```

```
Done
```

[25]:
```python
for i in range(1, 10):
    if i == 5:
        break
    print(i)
else:
    print('done')
```

```
1
2
3
4
```

[26]:
```python
for c in 'hello':
    print(c, end=' ')
```

```
h e l l o
```

2 Kattis problems

- almost every problem involves some form of loops!
- problems you may be able to solve from concepts learned so far:
- Oddities - https://open.kattis.com/problems/oddities
- A Terrible Fortress - https://open.kattis.com/problems/aterriblefortress
- Odd Echo - https://open.kattis.com/problems/oddecho
- FizzBuzz - https://open.kattis.com/problems/fizzbuzz
- Jumbo Javelin - https://open.kattis.com/problems/jumbojavelin
- Rating Problems - https://open.kattis.com/problems/ratingproblems
- Quality-Adjust Life-Year - https://open.kattis.com/problems/qaly
- File Extension - https://open.kattis.com/problems/nafnauki

[ ]:
```python
```