Lecture 5.3 : Miscellaneous

Introduction

- We review some previously met and present some new Python functions/objects.
- These may be useful to you when solving programming exercises.

`range`

- `range` can be useful when you need to generate some integers.
- Use `range(stop)` to generate integers `[0-(stop-1)]`.

```
print(list(range(10)))
```

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

- Use `range(start, stop[, step])` to generate integers `[start, start+step, start+2*step, ..., stop)` (up to but not including `stop`).

```
print(list(range(0, 10, 1))) # equivalent to range(10)
print(list(range(-5, 5, 2)))
print(list(range(5, -5, -2)))
```

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

`for` loops

- Use a `for` loop when you need to work your way across an iterable collection.

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

`break` and `continue`

- Use `break` to exit a loop early (perhaps an answer has been found so there's no point in going further).

```
for i in range(10):
    if i == 5: # we're done
        break
    print(i)
```

```
0
1
2
3
4
```

- Use `continue` to skip to the next iteration of a loop.

```
for i in range(10):
    if i % 2: # skip odd numbers
        continue
    print(i)
```

```
0
2
4
6
8
```

`zip`

- Use `zip` to join corresponding elements of iterables into a tuple.

```
numbers = [1, 2, 3, 4, 5]
words = ['one', 'two', 'three', 'four', 'five']

print(list(zip(numbers, words)))

for n, w in zip(numbers, words):
    print(f'{n} ---> {w}')
```

```
[(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four'), (5, 'five')]
1 ---> one
2 ---> two
3 ---> three
4 ---> four
5 ---> five
```

enumerate

- Use `enumerate` to associate an index with each element of an iterable yielding a tuple.

```
animals = ['penguins', 'lions', 'snakes']
print(list(enumerate(animals)))

for i, animal in enumerate(animals):
   print(f'At index {i} we find {animal}')
```

```
[(0, 'penguins'), (1, 'lions'), (2, 'snakes')]
At index 0 we find penguins
At index 1 we find lions
At index 2 we find snakes
```

sorted

- The `sorted` function does not work only on lists, it works on any **iterable**.
- Here we sort the characters in a string.

```
s = 'efdcgba'
print(sorted(s))
print(''.join(sorted(s)))
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
abcdefg
```

- We can also sort items in descending order.

```
s = 'efdcgba'
print(sorted(s, reverse=True))
print(''.join(sorted(s)))
```

```
['g', 'f', 'e', 'd', 'c', 'b', 'a']
abcdefg
```

- By specifying a `key` we can sort on arbitrary item attributes.

```
animals = ['ant', 'aardvark', 'tarantula', 'snake']

print(sorted(animals))
print(sorted(animals, key=len))
```

```
['aardvark', 'ant', 'snake', 'tarantula']
['ant', 'snake', 'aardvark', 'tarantula']
```

- The `key` does not have to be a built-in function.

```
def tagger(s):
   return s.count('t')

animals = ['ant', 'aardvark', 'tarantula', 'snake']

print(sorted(animals, key=tagger))
```

```
['aardvark', 'snake', 'ant', 'tarantula']
```

Random numbers

- In order to test our code or to run simulations we will often find it useful to generate *random* numbers.
- Python provides a `random` module which defines a number of useful methods in this regard.
- The `random()` method returns a random floating point number in the interval [0, 1). (This means 0 is included in the interval but 1 is not.)

```
from random import random

help(random)
```

```
Help on built-in function random:

random() method of random.Random instance
    random() -> x in the interval [0, 1).
```

```
from random import random

for i in range(3):
   print(f'{random():.2f}')
```

```
0.23
0.89
0.96
```

- The sequence appears random because the next number in the sequence cannot be predicted from previous ones.
- However the generated sequence is entirely determined by the initial *seed* supplied to the underlying algorithm.

- Seeding the generator with the same number causes the same sequence to be produced.
- Such generators are therefore referred to as *pseudo random number generators* (PRNGs).

```
from random import seed, random

seed(5)
for i in range(3):
   print(f'{random():.2f}')
```

```
0.62
0.74
0.80
```

```
from random import seed, random

seed(99)
for i in range(3):
   print(f'{random():.2f}')
```

```
0.40
0.20
0.18
```

```
from random import seed, random

seed(5)
for i in range(3):
   print(f'{random():.2f}')
```

```
0.62
0.74
0.80
```

- If we pass no argument to `seed` the PRNG is seeded with the current clock value. This provides enough randomness for most purposes.

Other `random` methods

- `randint(a,b)` generates a random integer in the range `[a, b]`.

```
from random import randint

help(randint)
```

```
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

```
for i in range(3):
   print(randint(10, 20))
```

```
20
18
10
```

- `choice(sequence)` returns a random element of *sequence*.

```
from random import choice

help(choice)
```

```
Help on method choice in module random:

choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

```
animals = ['llama', 'scorpion', 'bunny']

print(f'My favourite animal is the {choice(animals)}.')
```

```
My favourite animal is the scorpion.
```

- `shuffle(sequence)` shuffles the order of the elements of *sequence* in place (useful for generating permutations of the elements of a sequence).

```
from random import shuffle

help(shuffle)
```

```
Help on method shuffle in module random:

shuffle(x, random=None) method of random.Random instance
    Shuffle list x in place, and return None.

    Optional argument random is a 0-argument function returning a
    random float in [0.0, 1.0); if it is the default None, the
    standard random.random will be used.
```

```
animals = ['llama', 'scorpion', 'bunny']
shuffle(animals)
print(animals)
```

```
['scorpion', 'bunny', 'llama']
```

- `sample(sequence, N)` returns a new sequence containing *N* randomly selected elements of *sequence*.

```
from random import sample

help(sample)
```

```
Help on method sample in module random:

sample(population, k) method of random.Random instance
    Chooses k unique random elements from a population sequence or set.

    Returns a new list containing elements from the population while
    leaving the original population unchanged.  The resulting list is
    in selection order so that all sub-slices will also be valid random
    samples.  This allows raffle winners (the sample) to be partitioned
    into grand prize and second place winners (the subslices).

    Members of the population need not be hashable or unique.  If the
    population contains repeats, then each occurrence is a possible
    selection in the sample.

    To choose a sample in a range of integers, use range as an argument.
    This is especially fast and space efficient for sampling from a
    large population:   sample(range(10000000), 60)
```

```
animals = ['llama', 'scorpion', 'bunny']

favs = sample(animals, 2)

print(f"My two favourite animals are the {' and ' .join(favs)}.")
```

```
My two favourite animals are the llama and bunny.
```