1.0 Introduction

HOPE U ENJOY!

Print

In [1]:
```
print('University of Nottingham Ningbo China')
```

```
University of Nottingham Ningbo China
```

Tab Completion

While entering expressions in the shell, pressing the Tab key will search the namespace for any variables (objects, functions, etc.) matching the characters you have typed so far:

In [2]:
```
an_apple = 27
an_example = 42
```

In [3]:
```
# an<tab>
```

In [4]:
```
b = [1, 2, 3]
```

In [5]:
```
# b.<tab> b.append
```

Introspection

Using a question mark (?) before or after a variable will display some general information about the object:

In [6]:
```
b.append?
```

1.1 Data Structure

Tuple

In [28]:
```
tup = 4, 5, 6
tup
```

Out[28]:
```
(4, 5, 6)
```

In [30]:
```
nested_tup = (4, 5, 6), (7, 8)
nested_tup
```

Out[30]:
```
((4, 5, 6), (7, 8))
```

In [31]:
```
tuple([4, 0, 2])
```

Out[31]:
```
(4, 0, 2)
```

In [33]:
```
tup = tuple('string')
tup
```

Out[33]:
```
('s', 't', 'r', 'i', 'n', 'g')
```

In [34]:
```
tup[0]
```

Out[34]:
```
's'
```

In [38]:
```
tup = tuple(['foo', [1, 2]])
tup
```

Out[38]:
```
('foo', [1, 2])
```

In [40]:
```
'foo', [1, 2]
```

Out[40]:
```
('foo', [1, 2])
```

In [43]:
```
('you', 'me') * 4
```

Out[43]:
```
('you', 'me', 'you', 'me', 'you', 'me', 'you', 'me')
```

Unpacking tuples

If you try to assign to a tuple-like expression of variables, Python will attempt to unpack the value on the righthand side of the equals sign:

In [44]:
```
tup = (4, 5, 6)
```

In [45]:
```
a, b, c = tup
```

In [46]:
```
b
```

Out[46]:
```
5
```

In [53]:
```
x, *rest = tup
```

In [54]:
```
rest # return list
```

Out[54]:
```
[5, 6]
```

swap

In [47]:
```
a, b = 1, 2
```

In [48]:
```
a
```

Out[48]:
```
1
```

In [49]:
```
b
```

Out[49]:
```
2
```

In [50]:
```
b, a = a, b
```

In [51]:
```
a
```

Out[51]:
```
2
```

In [52]:
```
b
```

Out[52]:
```
1
```

List

In [56]:
```
a_list = ['foo', 'bar', 'baz']
```

In [57]:
```
a_list[1]
```

Out[57]:
```
'bar'
```

In [61]:
```
a_list.append('dwarf')
```

In [62]:
```
a_list
```

Out[62]:
```
['foo', 'bar', 'baz', 'dwarf']
```

In [63]:
```
a_list.insert?
```

In [64]:
```
a_list.insert(1, 'red')
```

In [65]:
```
a_list
```

Out[65]:
```
['foo', 'red', 'bar', 'baz', 'dwarf']
```

In [66]:
```
'dwarf' in a_list
```

Out[66]:
```
True
```

In [68]:
```
'dwarf' not in a_list
```

Out[68]:
```
False
```

Combine list

In [69]:
```
[4, 'foo'] + [7, 8, (2, 3)]
```

Out[69]:
```
[4, None, 'foo', 7, 8, (2, 3)]
```

In [70]:
```
x = [4, 'foo']
x.extend([7, 8, (2, 3)])
```

In [71]:
```
x
```

Out[71]:
```
[4, 'foo', 7, 8, (2, 3)]
```

sort

In [72]:
```
a = [7, 2, 5, 1, 3]
a.sort()
```

In [73]:
```
a
```

Out[73]:
```
[1, 2, 3, 5, 7]
```

In [74]:
```
b = ['saw', 'small', 'He', 'foxes', 'six']
b.sort(key=len)
```

In [75]:
```
b
```

Out[75]:
```
['He', 'saw', 'six', 'small', 'foxes']
```

Slicing (important!)

In [76]:
```
seq = [7, 2, 3, 7, 5, 6, 0, 1]
```

In [77]:
```
seq[1:5]
```

Out[77]:
```
[2, 3, 7, 5]
```

In [78]:
```
seq[:5]
```

Out[78]:
```
[7, 2, 3, 7, 5]
```

In [79]:
```
seq[3:]
```

Out[79]:
```
[7, 5, 6, 0, 1]
```

In [80]:
```
seq[:-2]
```

Out[80]:
```
[7, 2, 3, 7, 5, 6]
```

In [81]:
```
seq[::2]
```

Out[81]:
```
[7, 3, 5, 0]
```

In [84]:
```
seq[::-1]
```

Out[84]:
```
[1, 0, 6, 5, 7, 3, 2, 7]
```

Built-in Sequence Function

In [88]:
```
some_list = [1, 2, 3]
```

In [89]:
```
for value in some_list:
    print(value)
```

```
1
2
3
```

Dictionary

In [91]:
```
d = {'a': 'some value', 'b': [1, 2, 3, 4]}
```

In [92]:
```
d
```

Out[92]:
```
{'a': 'some value', 'b': [1, 2, 3, 4]}
```

In [99]:
```
d['a']
```

Out[99]:
```
'some value'
```

Set

In [101]:
```
set([2, 2, 2, 1, 3, 3])
```

Out[101]:
```
{1, 2, 3}
```

In [102]:
```
a = {1, 2, 3, 4 ,5}
b = {3, 4, 5, 6, 7, 8}
```

In [103]:
```
a & b
```

Out[103]:
```
{3, 4, 5}
```

List, Set, and Dict Comprehensions

In [104]:
```
a_list = ['a', 'as', 'bat', 'car', 'dove', 'python']
```

In [115]:
```
# step by step

ans = []
for x in a_list:
    if len(x) > 2:
        ans.append(x.upper())

ans
```

Out[115]:
```
['BAT', 'CAR', 'DOVE', 'PYTHON']
```

In [116]:
```
[x.upper() for x in a_list]
```

Out[116]:
```
['A', 'AS', 'BAT', 'CAR', 'DOVE', 'PYTHON']
```

In [117]:
```
[x.upper() for x in a_list if len(x) > 2]
```

Out[117]:
```
['BAT', 'CAR', 'DOVE', 'PYTHON']
```

1.2 Function

In [302]:
```
def my_function(x, y, z=1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

In [303]:
```
my_function(3.14, 7, 3.5)
```

Out[303]: 35.49

In [304]:
```
my_function(10, 20)
```

Out[304]: 45.0

In [305]:
```
my_function(5, 6, z=0.7)
```

Out[305]: 0.06363636363636363

In [306]:
```
my_function(y=6, x=5, z=7)
```

Out[306]: 77

Namespaces, Scope, and Local Functions

In [313]:
```
test = 1

def afunc():
    # global test
    test = 2

afunc()
```

In [314]:
```
test
```

Out[314]: 1

Lambda

In [315]:
```
f = lambda x: x * 2
```

In [317]:
```
f(4)
```

Out[317]: 8

In [319]:
```
def apply_to_list(a_list, f):
    return [f(x) for x in a_list]
```

In [320]:
```
ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x * 2)
```

Out[320]:
```
[8, 0, 2, 10, 12]
```

For

In [321]:
```
for i in range(10):
    print(i)
```

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

In [ ]:
```
count = 1
count
```

In [ ]:
```
num = 2
while num > 1:
    if num % 2 == 1:
        num = num * 3 + 1
    else:
        num /= 2
```

1.3 汉诺塔

In [292]:
```
def en_to_num(en):
    delta = ord('A') - ord('0')
    return int(chr(ord(en) - delta))
```

In [297]:
```
def move_tower(rod_x, rod_y):
    global count, tower

    print(rod_x, '->', rod_y)
    int_x = en_to_num(rod_x)
    int_y = en_to_num(rod_y)

    subject = tower[int_x][0]
    tower[int_x].pop(0)
    tower[int_y].insert(0, subject)
    print(tower)
    print()

    count += 1
    return
```

In [298]:
```
def rec(n, a, b_temp, c):
    if n == 1:
        move_tower(a, c)
        return
    rec(n - 1, a, c, b_temp)
    rec(1, a, b_temp, c)
    rec(n - 1, b_temp, a, c)
    return
```

In [299]:
```
n = 3
count = 0
tower = [list(range(n)), [], []]
rec(n, 'A', 'B', 'C')
```

```
A -> C
[[1, 2], [], [0]]

A -> B
[[2], [1], [0]]

C -> B
[[2], [0, 1], []]

A -> C
[[], [0, 1], [2]]

B -> A
[[0], [1], [2]]

B -> C
[[0], [], [1, 2]]

A -> C
[[], [], [0, 1, 2]]
```

In [296]:
```
count
```

Out[296]: 7

In [ ]:
```

```