Difference between Python v2.x and v3.x

This notebook documents the major differences between Python v2.x and v3.x relevant to our class.

According to python.org (https://wiki.python.org/moin/Python2orPython3):

> *Python 2.x is legacy, Python 3.x is the present and future of the language*

So in our class, we will use v3.x. However, they are legacy codes and computer systems where v2.x may still exist and is needed.

The following is a list of differences to be discussed in this notebook:

- "print" function
- Division operator
- xrange
- Error handling

1. The "print" function

The **print** function in Python v2.x has been replaced by **print()** in Python v3.x, i.e., a pair of parenthesis is needed. In other words, **print** function with and without parenthesis will work in both Python v2.x and v3.x. However, in Python v3.x, you can only use **print()**.

In [1]:
```
print 'Hello, world!'   #Works with Python 2.x but not 3.x
```

```
Hello, world!
```

In [2]:
```
print('Hellow, world!') #Works with both Python 2.x and 3.x
```

```
Hellow, world!
```

2. Division operator

In Python v2.x, the "/" operator works as a floor division for integers. However, it returns a float value if at least one of the arguments is a float. In Python v3.x, the "/" operator does floating point division for both integer and float arguments.

In [3]:
```
print(3/2)   #will produce 1 in Python v2.7
print(3.0/2) #will produce 1.5 in Python v2.x and v3.x
```

```
1
1.5
```

In Python, the real floor division operator is "//". It returns the floor value for both interger and floating point arugments. This operator is the same in both v2.x and v3.x.

In [4]:
```
print(3//2)
print(3.0//2.0)
```

```
1
1.0
```

3. xrange

The function "xrange()" is valid in Python v2.x but obsolete in Python v3.x. Instead, Python v3.x uses "range()" for the same function of "xrange()".

In Python, the function "range()" creates a list. The difference between "range()" and "xrange()" is related to how the list is allocated and used in memory. If you are not dealing with large amount of data, it is not critical.

In [5]:
```
print(range(0,5))    #works in both Python v2.x and v3.x
print(xrange(0,5))   #only works in Python v2.x, but not v3.x
```

```
[0, 1, 2, 3, 4]
xrange(5)
```

4. Error handling

In Python, you can try to catch exceptions (errors) and deal with them in your code. In Python v3.x, an "as" is required. See the difference in the following examples.

In [6]:
```
#This is how it should be in Python v3.x. It works in both v2.x and v3.x.
try:
    print(3/0)
except Exception as e:
    print("type error: " + str(e))
```

```
type error: integer division or modulo by zero
```

In [7]:
```
#This will work in Python v2.x, but not in v3.x.
try:
    print(3/0)
except Exception, e:
    print("type error: " + str(e))
```

```
type error: integer division or modulo by zero
```

How to check the version of Python you are using?

Because of all these differences in the versions of Python, and to ensure your Python code works correctly, it is sometime necessary to require a minimum version of Python. In this case, you can use the **sys** module. For example, put the following at the beginning of your code will require a Python version of 3.7 or later.

In [8]:
```
import sys

print('Current Python version information:', sys.version)

if sys.version_info<(3,7,0):
  sys.stderr.write("You need python 3.7 or later to run this script\n")
  exit(1)
```

```
('Current Python version information:', '2.7.13 |Anaconda 4.3.0 (64-bit)| (defau
lt, Dec 19 2016, 13:29:36) [MSC v.1500 64 bit (AMD64)]')

You need python 3.7 or later to run this script
```

If you are in command line environment, you can type:

```
$ python -V
```

to check the version of Python.