- **Differences between 2.X and 3.Xpacakages**
    - Division operator
    - print function
    - Unicode
    - xrange
    - Error Handling
    - _future_ module

Division operator
If we are porting our code or executing python 3.x code in python 2.x, it can be dangerous if integer division changes go unnoticed (since it doesn't raise any error). It is preferred to use the floating value (like 7.0/5 or 7/5.0) to get the expected result when porting our code.
Example:

```
print 7 / 5

print -7 / 5
```

Output in Python 2.x :

```
1

-2
```

Output in Python 3.x :

```
1.4

-1.4
```

print function
This is the most well-known change. In this, the print keyword in Python 2.x is replaced by the print() function in Python 3.x. However, parentheses work in Python 2 if a space is added after print keyword because the interpreter evaluates it as an expression.
Example:

```
print 'hello naveen'          #python 3.x doesn't support this statement

print("Hello naveen")
```

Unicode:
In Python 2, implicit str type is ASCII. But in Python 3.x implicit str type is Unicode.
Example:

```
print(type('default string '))

print(type(b'string with b '))
```

Output in Python 2.x (Bytes is same as str)

```
<type 'str'>

<type 'str'>
```

Output in Python 3.x (Bytes and str are different)

```
<class 'str'>

<class 'bytes'>
```

Python 2.x also supports Unicode

```
print(type('default string '))

print(type(u'string with b '))
```

Output in Python 2.x (Unicode and str are different)

```
<type 'str'>

<type 'unicode'>
```

Output in Python 3.x (Unicode and str are same)

```
<class 'str'>

<class 'str'>
```

xrange:
xrange() of Python 2.x doesn’t exist in Python 3.x. In Python 2.x, range returns a list i.e. range(3) returns [0, 1, 2] while xrange returns a xrange object i. e., xrange(3) returns iterator object which work similar to Java iterator and generates number when needed.

Example:

```
for x in xrange(1, 5):                #python 3.x doesn't support xrange()
    print(x)

for x in range(1, 5):
    print(x)
```

Output in Python 2.x

1 2 3 4 1 2 3 4

Output in Python 3.x

NameError: name 'xrange' is not defined

Error Handling:
There is a small change in error handling in both versions. In python 3.x, ‘as’ keyword is required**.**
Example:

```
try:

    trying_to_check_error

except NameError, err:

    print err, 'Error Caused'   # Would not work in Python 3.x



'''
```

Output in Python 2.x:

name 'trying_to_check_error' is not defined Error Caused

Output in Python 3.x :

```
File "a.py", line 3
    except NameError, err:
                    ^
SyntaxError: invalid syntax
```

Example2:

```
try:
    trying_to_check_error
except NameError as err: # 'as' is needed in Python 3.x
    print (err, 'Error Caused')

'''
```

Output in Python 2.x:

```
(NameError("name 'trying_to_check_error' is not defined",), 'Error Caused')
```

Output in Python 3.x :

```
name 'trying_to_check_error' is not defined Error Caused
'''
```

__future__ module:

This is basically not a difference between the two versions, but a useful thing to mention here. The idea of __future__ module is to help migrate to Python 3.x.
If we are planning to have Python 3.x support in our 2.x code, we can use _future_ imports in our code.
For example, in the Python 2.x code below, we use Python 3.x's integer division behavior using the __future__ module.

Example:

```
# In below python 2.x code, division works
# same as Python 3.x because we use  __future__

from __future__ import division

print 7 / 5

print -7 / 5
```

output:
1.4
-1.4