Python 2 vs. Python 3 for Carleton CS students

Following are the differences between Python 2 and Python 3 that are likely to come up for students who learned Python 3 in CS 111 and are now using Python 2 in another class, or for students learning Python 2 in CS 111 but reading the Zelle textbook which uses code examples in Python 3. For a more complete explanation of all of the updates in Python 3, see this page in the Python documentation.

0. The quick summary

<table>
<thead>
<tr>
<th>Python 2</th>
<th>Python 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print x</code></td>
<td><code>print(x)</code></td>
</tr>
<tr>
<td><code>4/3 = 1</code></td>
<td><code>4/3 = 1.33333 4//3 = 1</code></td>
</tr>
<tr>
<td><code>raw_input()</code></td>
<td><code>input()</code></td>
</tr>
<tr>
<td><code>file(“my_file.txt”)</code></td>
<td><code>open(“my_file.txt”)</code></td>
</tr>
<tr>
<td><code>xrange()</code></td>
<td><code>range()</code></td>
</tr>
</tbody>
</table>

1. The print statement/function

The print statement in Python 2 becomes a print() function in Python 3.

- For basic print functionality the only difference is whether or not to use parentheses

  ```python
  Python 2:  print "The answer is", 42
  Python 3:  print("The answer is", 42)
  Output:  The answer is 42
  ```

  ```python
  Python 2:  print
  Python 3:  print()
  Output: newline
  ```

- To format printed output, Python 2 uses special syntax while Python 3 uses the keyword arguments `sep` and `end`. `sep` determines the separator used between arguments to the print function (default is space), and `end` determines the final character printed (default is newline)

  ```python
  Python 2:  print "The answer is",  # comma suppresses newline
              print 42
  Python 3:  print("The answer is", end=" ")
              print(42)
  Output:  The answer is 42
  ```

  ```python
  Python 3:  print(“01”, “12”, “1981”, sep=”-“)
  Output:  01-12-1981
  ```
If you are familiar with the print() function in Python 3, you can still choose to use it when coding in Python 2 by using the __future__ module.

```python
from future import print_function
```

2. Division
int/int always returns and int in Python 2, truncating the result if it’s not a whole number. In order to get a float result from division you must have at least one float argument. int/int always returns a float in Python 3, even if the result is a whole number. In Python 3 int//int always returns an int, truncating the result if it’s not a whole number, in the same way a single / works in Python 2.

| Python 2: 4/3 | # result is 0 |
| Python 2: 3/3 | # result is 1 |
| Python 2: 4.0/3 | # result is 1.33333 |
| Python 2: 3.0/3 | # result is 1.0 |
| Python 3: 4/3 | # result is 1.33333 |
| Python 3: 3/3 | # result is 1.0 |
| Python 3: 4//3 | # result is 1 |
| Python 3: 3//3 | # result is 1 |

Once again you can use the division operator from Python 3 in Python 2 by importing it from the __future__ module.

```python
from future import division
```

3. Input
The raw_input() function in Python 2 is equivalent to input() in Python 3. These functions always return user input as a STRING, which must be converted if you want a different type. In the Zelle Python 3 textbook you will often see eval(input()) as a method to get user input as something other than a string, however you SHOULD NOT use this. EVER. Or at least in this class. Instead you should convert the input to the exact type you wish.

| Python 2: the_input = raw_input()  | # the_input is of type string  |
| Python 3: the_input = input()     | # the_input is of type string  |
| Python 2: the_input = float(raw_input()) | # the_input is of type float |
| Python 3: the_input = float(input())  | # the_input is of type float |
| Python 2: the_input = int(raw_input()) | # the_input is of type int |
| Python 3: the_input = int(input())   | # the_input is of type int |
| Zelle: the_input = eval(input())    | # DON'T USE |
4. Files
The `file` command in Python 2 is removed in Python 3, you have to use the `open()` function instead.

```python
Python 2:
for line in file("my_file.txt"):  
    print line

Python 3:
myFile = open("my_file.txt",'r'):  
for line in myFile:  
    print line
```

5. Range
The `range()` function in Python 3 is like `xrange()` in Python 2, it does not return a list and can handle an arbitrarily large value.

```python
Python 2:  L = range(10)  # L is [0,1,2,3,4,5,6,7,8,9]
Python 3:  L = list(range(10))  # L is [0,1,2,3,4,5,6,7,8,9]

# The following causes an error in Python 2 but is valid in Python 3
for i in range(10000000000000):
```