Python Version Changes

Version 2 vs. Version 3

Some Changes in Python Version 3

- raw_input is replaced by input.
- Sorts – key words – cmp key word removed
  - Only key and reverse remain
  - Sort ordering has changed significantly
- print statement versus function
- range, zip and dictionary unload methods
- Integer division and types
- Comparing numerics and non-numerics
- Text representation – simplified
  - Translate/maketrans major changes
- Class types and designations
- Python 2to3 tool helps in conversions

Print

- The new print function replaces the print statement/function from 2.x.
- There are new key words that allow easy modification of behavior.
- sep=
  - Controls the space between items being printed without explicit formatting.
  - The default is the same as before; a space
  - Vailid entries include a null string as well as a multi-character string.
- end=
  - The default is the same as before; a newline character (\n).
  - Any replacement that does not include a \n will suppress linefeeds.
- file=
  - Printing can be easily directed to any valid opened file.
  - The default is still the stdout which is usually your screen.
- Sample on next slide

Print Function

```
x = 1
y = 2

print('I have 1 item for 2 people')
print('I have', x, 'item for', y, 'people')
print('I have', x, 'item for', y, 'people', sep = '')
print('I have', x, 'item for', y, 'people', sep = '-*-')

print('I have', x, 'item for', y, 'people')
print('What should I do?')

print('I have', x, 'item for', y, 'people.', end = ' ')
print('What should I do?')
```

```
I have 1 item for 2 people
I have 1 item for 2 people
I have1item for2people
I have-*-1-*-item for-*-2-*-people
I have 1 item for 2 people
What should I do?
I have 1 item for 2 people. What should I do?
```

Iterators Replace Lists

- In Python 2 many of the functions/methods produced lists.
- In Python 3 some of these have become iterators.
- range
  - The range function now produces an iterator. xrange has been eliminated.
  - To produce a list, wrap range in a list function: e.g.; list(range(10))
- zip - zipping two iterables together now produces an iterator, not a list of tuples.
- Dictionary unloads don't really unload
  - Dictionary methods keys, values and items now produce views.
  - As with range, use the list function to produce a list if needed.
  - iterkeys, itervalues and iteritems have been removed.
- The sorted and reversed functions still produce the same results.

Integer Division and Types

- Integer division in Python version 2
  - >>> 7/3
    2
- Integer division in Python version 3
  - >>> 7/3
    2.333333333333333
- Types in Python version 2
  - >>> x = 12.3
    >>> type(x)
    <type 'float'>
- Types in Python version 3
  - >>> x = 12.3
    >>> type(x)
    <class 'float'>

Comparing Numerics and Strings

- Usually, this is done in error and results in a semantic error in Python 2
- Python 3 has changed this result somewhat as shown in the shell operations below.
- Objects of different types except numbers are ordered by their type names.

Python Version 2

```
>>> x = 'a'
>>> x == 1
False
>>> x < 1
False
>>> x > 1
True
>>> x > 99999999999999
True
```

Python Version 3

```
>>> x = 'a'
>>> x == 1
False
>>> x > 1
Traceback (most recent call last):
TypeError: unorderable types: str() > int()
>>> x < 1
Traceback (most recent call last):
TypeError: unorderable types: str() < int()
```

Sort Ordering

- In Python V2, you could sort items containing both numerics and non-numerics:

```
y = ['text', 2, True, None]; y.sort()  # An extreme example
result - [None, True, 2, 'text']
```

- In Python V3, this unexpected result is no longer supported:

```
y = ['text', 2, True, None]; y.sort()  # An extreme example
result -   Traceback (most recent call last):
           File "<pyshell#0>", line 1, in <module>
           y = ['text', 2, True, None]; y.sort()
           TypeError: unorderable types: int() < str()  or
           TypeError: unorderable types: int() < NoneType() or
           TypeError: unorderable types: str() < NoneType()
```

- Booleans are still considered numbers (True = 1.0, False = 0.0).
  - They will sort just fine with other numbers.

Classes

- In Python 2 there were two ways to define a class:
  - class classname(); pass # creates a classic-style class
  - class classname(object); pass # creates a new-style class
- In Python 3:
  - Classic-style classes have been removed
  - Both of the above definitions create a new-style class
  - If you don't place "object" inside the parentheses. The parentheses can be removed. (e.g.; class classname)

Text Representation

- Python 2 assumed you were using ASCII while Python 3 assumes Unicode.
- For programmers who are dealing exclusively with ASCII, the change is minor.
- There are some new string methods covered in Python 3 Notes.
- translate/maketrans operation has changed completely
  - the 2to3 utility does not address this change at all

translate and maketrans

- In Python 2.x translate/maketrans operated this way:
  - You had to import maketrans from the string module.
  - maketrans took two arguments:
    - The characters you wanted to translate.
    - The corresponding characters to replace the first characters.
    - With these two arguments maketrans would create a 256-byte table corresponding to every possible character a byte can hold.
  - translate used one or two arguments:
    - First argument: the maketrans result to translate a given string.
    - Second argument: if supplied, these characters were removed from the string.

translate and maketrans

- In Python 3.x translate/maketrans operate this way:
  - maketrans is now a built-in static method – no need to import.
  - maketrans takes two or three arguments:
    - The first two arguments are the same ones previously used in maketrans.
    - The third argument (optional) contains the characters to be deleted.
    - With these arguments maketrans creates a dictionary of the characters to be translated and/or deleted.
  - translate uses the result of maketrans to process the string in question. Example:

    x = "Don't worry - be happy!"

    y = x.translate(str.maketrans('aeiou', '12345', "-"))

    result in variable y - D4n't w4rry b2 h1ppy!

Conversion Assistance

- When working in Python 2.x, use the __future__ module
  - from __future__ import division, print_function
  - integer division and the print function will now operate as they do in Python 3.x
- Using the 2to3 module – use 2to3 as the command
  - Ex: 2to3 –w /home/student/[progname.py]
  - Processes one program or an entire directory
  - Windows must use:
    python c:\full path\2to3.py file_dir
  - For details go to this link.