For performance, especially when you're iterating over a large range, xrange() is usually better. However, there are still a few cases why you might prefer range():

* In python 3, range() does what xrange() used to do and xrange() does not exist. If you want to write code that will run on both Python 2 and Python 3, you can't use xrange().
* range() can actually be faster in some cases - eg. if iterating over the same sequence multiple times. xrange() has to reconstruct the integer object every time, but range() will have real integer objects. (It will always perform worse in terms of memory however)
* xrange() isn't usable in all cases where a real list is needed. For instance, it doesn't support slices, or any list methods.

[Edit] There are a couple of posts mentioning how range() will be upgraded by the 2to3 tool. For the record, here's the output of running the tool on some sample usages of range() and xrange()

RefactoringTool: Skipping implicit fixer: buffer

RefactoringTool: Skipping implicit fixer: idioms

RefactoringTool: Skipping implicit fixer: ws\_comma

--- range\_test.py (original)

+++ range\_test.py (refactored)

@@ -1,7 +1,7 @@

for x in range(20):

- a=range(20)

+ a=list(range(20))

b=list(range(20))

c=[x for x in range(20)]

d=(x for x in range(20))

- e=xrange(20)

+ e=range(20)

As you can see, when used in a for loop or comprehension, or where already wrapped with list(), range is left unchanged.