1. **Python 3.x vs. 2.x Changes:**

- The `print()` statement has been replaced with a `print()` function, with keyword arguments to replace most of the special syntax of the old print statement. New function syntax:
  
  ```python
  print(value,...,sep=' ',end='
', file=sys.stdout)
  ```

a) Predict the expected output of each of the following.

<table>
<thead>
<tr>
<th>Version 2.x</th>
<th>Version 3.x</th>
<th>Expected Output</th>
</tr>
</thead>
</table>
| `print ('cat',5,'dog')`          | `print ('cat',5,'dog')`          | `cat55dog
`               |
| `print()`                        | `print()`                        | `n`                     |
| `print ('cat',5)
print 'horse'
print 'cow'`                    | `print ('cat',5,end='')`
```
| `print ('horse')`
`print ('cow')`               | `print ('horse')`                | `cow
`                 |

Version 3.x

<table>
<thead>
<tr>
<th></th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print ('cat',5,'dog',sep='23',end='#')</code></td>
<td><code>cat23523dog#</code></td>
</tr>
<tr>
<td><code>print ('cat',5,'dog',end='!',sep='23')</code></td>
<td><code>cat235!dog#</code></td>
</tr>
<tr>
<td><code>print ('cat',5,'dog',sep='23','horse')</code></td>
<td><code>error</code></td>
</tr>
</tbody>
</table>
| `print ('cat',5,'dog',sep='>!*3)`    | `cat>5>7>dog
`         |

- The `range()` now behaves like `xrange()` of version 2.x. The `xrange()` function no longer exists in version 3.
- `raw_input()` was renamed to `input()`. That is, the new `input()` function reads a line from `sys.stdin` and returns it as a string with the trailing newline stripped. It raises `EOFError` if the input is terminated prematurely. To get the old behavior of `input()`, use `eval(input())`. Example, use a `for` loop to generate a sequence of values one at a time for each iteration of the loop:
  
  ```python
  n = eval(input("Enter # of iterations? "))
  for count in range(n):
    print(count, end=" ")
  print("\nDone")
  ```

```
Enter # of iterations? 6
0 1 2 3 4 5
Done
```

- Removed `<>` as an alternate “not equal” operator, so use `!=` instead.
- There is only one built-in integral type, named `int`. It behaves like the old `long` type.
- An expression like `1/2` returns a float. Use `1//2` to get the truncating “integer division” behavior of version 2.
- Dictionary methods `dict.keys()`, `dict.items()` and `dict.values()` return iterable “views” instead of lists. For example, this no longer works: `keyList = d.keys(); keyList.sort()`. Use `keyList = sorted(d)` instead.
(Also, the `dict.iterkeys()`, `dict.iteritems()` and `dict.itervalues()` methods are no longer supported.)

2. Review of assignment statements. Predict the output of the following programs

```python
a = 123
b = a
a += 1
print ('a is', a)
print ('b is', b)
print()
c = ['cat', 'dog']
d = c
c.append('cow')
print('c is', c)
print('d is', d)
```

```
a is 124
b is 123

[c, 'dog', 'cow']
d is [c, 'dog', 'cow']
```

```
c = 'cat'
d = c
c += 'fish'
print('c is', c)
print('d is', d)
```

```
c is catfish
d is cat
```

```
a = 123
```

```
9
123
124
```

```
```
Lecture 1 Page 1
3. Design a program to roll two 6-sided dice 1,000 times to determine the percentage of each outcome (i.e., sum of both dice). Report the outcome(s) with the highest percentage.

Most simple programs have a similar functional-decomposition design pattern:

```
import random

random.randint(1, 6)

outcome_counts = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

a) Customize the diagram for the dice problem by briefly describing what each function does and what parameters are passed.

b) An alternative design methodology is to use object-oriented design. For the above dice problem, what objects would be useful and what methods (operations on the objects) should each perform?

Objects: Die with methods: roll, get Roll, get Sides

TallySheet with methods: increment, clear,
import random

def main():
    numRolls, numSides, numDice = getInputs()
    outcomeCounts = rollDice(numRolls, numSides, numDice)
    outcomePercentages = calcPercentages(outcomeCounts, numRolls)

    def getInputs():
        numRolls = input("Enter # of rolls: ")
        numSides = 
        numDice = 
        return numRolls, numSides, numDice

    def rollDice(numRolls, numSides, numDice):
        outcomeCounts = [0] * (numSides * numDice + 1)
        for i:
            return outcomeCounts

    main()