CIS 210

Key:
- Binary representation/Binary $\leftrightarrow$ Decimal
- Cipher algorithm
- Exercises and Q/A
- Midterm exam

Recall:

```python
>>> twice
<function twice at 0x105b19b70>
```

```python
>>> id(twice)
4390493040
```
### CIS 210

#### Binary representation of numbers

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'0b1'</td>
</tr>
<tr>
<td>2</td>
<td>'0b10'</td>
</tr>
<tr>
<td>4</td>
<td>'0b100'</td>
</tr>
<tr>
<td>10</td>
<td>'0b1010'</td>
</tr>
<tr>
<td>32</td>
<td>'0b100000'</td>
</tr>
<tr>
<td>33</td>
<td>'0b100001'</td>
</tr>
</tbody>
</table>

469<sub>10</sub> = (4 * 10<sup>2</sup>) + (6 * 10<sup>1</sup>) + (9 * 10<sup>0</sup>)
Binary representation of numbers

CIS 210

Binary to decimal:

111010101₂ = (1 * 2⁸) + (1 * 2⁷) + (1 * 2⁶) + (0 * 2⁵) +
(1 * 2⁴) + (0 * 2³) + (1 * 2²) + (0 * 2¹) +
(1 * 2⁰)

- look at each bit in the binary number, b.
- multiply it by the correct power of 2
- sum the results

Decimal to binary:

469 ÷ 2 q234 r1 → rightmost bit
234 ÷ 2 q117 r0 → next bit
117 ÷ 2 q58 r1 → next bit
58 ÷ 2 q29 r0 → next bit
29 ÷ 2 q14 r1 → next bit
14 ÷ 2 q7 r0 → next bit
7 ÷ 2 q3 r1 → next bit
3 ÷ 2 q1 r1 → next bit
1 ÷ 2 q0 r1 → leftmost bit

the first quotient is the decimal number to be converted while quotient is greater than 0
- divide the decimal number by 2
- make the remainder the next digit to the left in the answer
- update the quotient
characters are also represented using binary ASCII -> UTF-8 standards

```python
>>> ord('Z')
90
>>> ord('a')
97
```

binary representation of characters - ASCII

```python
>>> ord('Z')
90
>>> chr(90)
'Z'
>>> ord('a')
97
>>> chr(97)
'a'
>>> for i in range(90, 97):
    print(chr(i))
```

characters are also represented using binary ASCII -> UTF-8 standards

```python
>>> ord('Z')
90
>>> bin(ord('Z'))
'0b1011010'
>>> ord('a')
97
>>> bin(ord('a'))
'0b1100001'
```

binary representation of characters

```python
>>> ord('Z')
90
>>> ord('a')
97
>>> 'a' < 'Z'
False
>>> 'a' < 'Z'.lower()
True
```
binary representation of characters
UTF-8

>>> '\u03a9'
'Ω'

>>> '\u2877'
'⡷'

Figure 3.2
Figure 3.5

symmetric key algorithm
functions can be composed

For example,

```python
>>> y = 4.5
>>> z = abs(round(y))
??
```

Monte Carlo Simulation (Problem 3-1)

```python
import random
import math

def montePi(numDarts):
    """
    inCircle = 0
    for i in range(numDarts):
        x = random.random()
        y = random.random()
        d = math.sqrt(x**2 + y**2)
        if d <= 1:
            inCircle += 1
    >>> montePi(10)
    ??
    pi = inCircle / numDarts * 4
    return pi
```
CIS 210

Python toolkit so far

- numeric data types (int, float) and operations (e.g., +, **, round, abs)
- string data type and operations (e.g., +, len, count, find, format)
- Boolean data type and operations (e.g., <, and)
- NoneType (None)
  expressions

- Python Standard Library – math, turtle, random modules; import
  assignment statement
- Python repetition – for, while
- Python conditionals – if

- variable assignment
- user-defined functions; function design; docstrings
- IDLE interactive development environment; help function

CIS 210

CIS 210 Learning Outcomes

- understand, develop, implement algorithms for computational problem solving;
- use structured design and testing methods to develop and implement programs;
- read, write, revise, document, test, and debug code;
- demonstrate robust mental models of data representation and code execution;
- demonstrate good understanding of a high level programming language;
- introduce and/or implement a sampling of classic computer science problem domains and algorithms.

CIS 210

Programming/Computer Science concepts

- Computational Problem Solving: designing, implementing, checking, revising algorithms/programs.
- Good programming style: function docstrings (type contract; description including parameters, returned value, and side effects if any; examples of function use), well-named variables, use of whitespace between operators and sections of code, judicious use of inline comments (why not what).
- Python primitive elements: Objects - value/attributes, operations, memory location (id).
  - Combining primitive elements: Expressions - expressions evaluate to a value; overloaded operators; methods/functions/operators; short circuit evaluation of boolean expressions
  - Naming values: Variables/assignment - assignment statements are not expressions and do not return a value; namespace; variable scope; dynamic typing; strong typing.
  - User defined functions – functions always return a value (sometimes None); functions sometimes have side effects.
- What happens when a function is called?
  - Activation record/frame added to call stack for local namespace; return address
  - Call-by-assignment parameter passing
  - Side effects/returning a value
- Iterative algorithms; accumulator pattern
- Monte Carlo algorithms