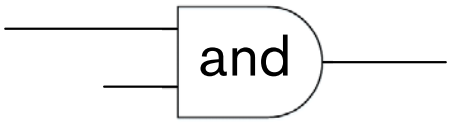
Coding in **Python**  Elements of Discrete Mathematics

Answers and Solutions

Maria Litvin
Phillips Academy, Andover, Massachusetts

Gary Litvin
Skylight Software, Inc.

Skylight Publishing
Andover, Massachusetts

Skylight Publishing
9 Bartlet Street, Suite 70
Andover, MA 01810

web: http://www.skylit.com
e-mail: sales@skylit.com
support@skylit.com

Library of Congress Control Number: 2019905086

ISBN 978-0-9972528-4-2

2 3 4 5 6 7 23 22 21

Printed in the United States of America

Contents

1 An Introduction to Computers and Coding in Python

Section 1.2

2. D. RAM stands for Random Access Memory; its bytes can be addressed in random order.

3. $\frac{120 \cdot 2^{30}}{512 \cdot 2^{20}} = 240$

5. There are 95 printable ASCII characters, so the 256 possible arrangements of the bits in one byte are sufficient. $2^7 = 128$, so 7 bits suffices.

6. This segment of code computes the sum of all the numbers from 1 to 6. `AX` contains 0007, since it increases by 1 until it is greater than 6. `BX` contains the sum, `0015`. This number represents 21 in the hexadecimal number system; 15 hex is $1 \cdot 16 + 5 = 21$.

Section 1.3

1. Redundancy is less than optimal representation or transmission of information, which makes it possible to interpret a message and correct errors even when the message is garbled.

5. The + operator, when applied to two strings, concatenates them, that is, combines them into one string.

7. `9-8*2+6` gives `-1`, because multiplication is applied before addition and subtraction. `(5-1)*(1+2)**3` gives `108` because expressions in parentheses are evaluated first, and power is applied before multiplication. Thus `(5-1)*(1+2)**3` $= 4 \cdot 3^3$. The precedence of operators is parentheses, then `**`, then `*` and `/` from left to right, then + and – from left to right.

8. `m%n` returns the remainder when `m` is divided by `n`. `%` is applied before + and –; it has the same rank as `*` and `/`.

Section 1.4

3. C

6. A

2 Variables and Arithmetic

Section 2.2

3. (a), (c), (e)

4. `def`, `for`, `in`, `return`

6.
```
One is better than
one; two is better than one
```

`\n` in the string is an escape character that means newline.

9. The line `return n**3` needs to be moved one space to the left for consistent indentation.

Section 2.3

1.
```
>>> r = 5
>>> pi = 3.14159
>>> area = pi * r**2  # or: area = pi * r * r
```

3. `name`, `d7`, `first_name`, `lastName`, `Amt`, and `LBS_IN_KG` are valid names for variables. `name`, `d7`, `first_name` and `LBS_IN_KG` are common Python style. `lastName` style, often called "lower camel case," is more commonly used in Java than in Python; Python's style with underscores is called "snake_case." `Amt` is not a good style for a variable, because it starts with a capital letter. But all caps in `LBS_IN_KG` is fine because it is a universal constant.

6.
```
5
3
```

Notice that `a = a+b` sets `a` to 5.

7.
```
def greeting():
    """Ask the user for their name and print a "Welcome to Python" greeting.
    """
    name = input('What is your name? ')
    print('Hello, ' + name + '. Welcome to Python!')
```

Section 2.4

2.
```
(x - 2)**3 + 3*x
```

5.

```
y = x*x
y = y*y
```

7. (b) The function returns `None`. To verify this, write

```
print(triangle('*'))
```

3 Sets and Functions

Section 3.2

1. 4 subsets: $\varnothing$, $\{a\}$, $\{b\}$, $\{a, b\}$

3. $f : \{1, 2, 3, 4, 5\} \to \{11, 12, 13, 14, 15\}$; $f(x) = x + 10$.

6. $\ldots; f(-3) = 0; f(-2) = 1; f(-1) = 2; f(0) = 0; f(1) = 1; f(2) = 2; f(3) = 0; \ldots$. In other words, $f(n)$ is the remainder when n is divided by 3, as in `n%3` in Python.

7. 27:

x	1	2	3
$f(x)$	a	a	a

x	1	2	3
$f(x)$	a	a	b

x	1	2	3
$f(x)$	a	a	c

x	1	2	3
$f(x)$	a	b	a

x	1	2	3
$f(x)$	a	b	b

x	1	2	3
$f(x)$	a	b	c

x	1	2	3
$f(x)$	a	c	a

x	1	2	3
$f(x)$	a	b	b

x	1	2	3
$f(x)$	a	b	c

x	1	2	3
$f(x)$	b	a	a

x	1	2	3
$f(x)$	b	a	b

And so on...

9. $f(P) =$ distance from the center of the circle to P.

11.

```
9 5
```

A set cannot include duplicate values; a list can.

13.

```
def mex(s):
   """Calculate the mex (minimum excludant) of the set s
      of non-negative integers.
   """
    n = 0
    while n in s:
        n += 1
    return n
```

Section 3.3

1. Domain: $\mathbb{R} - \{2\}$; range: $\mathbb{R} - \{0\}$.

3. Domain: $-1 \le x \le 1$; range: $0 \le y \le 1$

4. The domain has 900 elements; the range has 27 elements (1 through 27).

7. Two: $A \to B, B \to C, C \to A$ and $A \to C, C \to B, B \to A$

9. $f(g(1)) = f(0) = 1$; $g(f(1)) = g(-1) = -2$

Section 3.4

1.
```
def concat_strings(s1, s2):
    return s1 + ' ' + s2
```

3.
```
def right_justify(s, w):
    """Return s padded with spaces on the left."""
    return (w - len(s))*' ' + s
```

```
>>> right_justify('123', 5)
'  123'
>>> right_justify(5, '123')
TypeError: object of type 'int' has no len()
>>> right_justify('123')
TypeError: right_justify() missing 1 required positional argument: 'w'
>>> right_justify('12345', 3)
'12345'
```

5.
```
**
*
###
##
#
2
```

8.
```
len(0)            Exception
len('0')          Returns 1
len('''0''')      Returns 1
len('''''')       Returns 0
len([0])          Returns 1
len([])           Returns 0
len((0, 1))       Returns 2
len(range(0, 1))  Returns 1
```

Section 3.5

2.

```
>>> a
1
>>> b
2
```

`swap(x, y)` swaps copies of the arguments passed to it; the original arguments remain unchanged.

3. (a)

```
from math import sqrt

def solve_quadratic(a, b, c):
    """Return the roots of the equation ax^2+bx+c=0."""
    d = sqrt(b*b - 4*a*c)
    return (-b - d)/(2*a), (-b + d)/(2*a)
```

Section 3.6

1. 0 and `n-1`.

3.

```
def sum_digits(n):
    """Return the sum of the digits in n."""
    return sum(int(d) for d in str(n))
```

7.

```
[1, 2, 3, 4]
```

4 Algorithms and `while` and `for` Loops

Section 4.2

2.

```
Input: n
k ← 1
sum ← 0
while k ≤ n:
    sum ← sum + k
    k ← k + 1
Output: sum
```

3.

```
Input: m, n
quotient ← 0
while m ≥ n:
    m ← m - n
    quotient ← quotient + 1
remainder ← m
Output: quotient, remainder
```

Section 4.3

2.

```
def sum_alt_reciprocals(n):
    """Return 1 - 1/3 + 1/5 - ... up to 1/n."""
    k = 1
    sign = 1
    _sum = 0
    while k <= n:
        _sum += sign/k
        k += 2
        sign = -sign
    return _sum

from math import pi

print('pi =', pi)
for n in (1000, 10000, 100000):
    s = 4*sum_alt_reciprocals(n)
    print('n = {0:7d}: 4*sum = {1:15.13f} diff = {2:15.13f}'.format(n,
                                                                    s, abs(pi - s)))
```

3.

```
n_max = int(input('Enter a positive odd integer: '))
sum_n = 0
for n in range(1, n_max+1, 2):
    sum_n += n
    print(n, sum_n)
```

6.

```
n_max = -1
while n_max <= 0:
    s = input('Enter a positive integer: ')
    try:
        n_max = int(s)
    except ValueError:
        print('Invalid input')

print() # print a blank line

n = 1
sum1n = 0
sum1n2 = 0
while n <= n_max:
    sum1n += n
    sum1n2 += n*n
    print('{0:3d} {1:6d} {2:6d} {3:6g}'.format(n, sum1n, sum1n2, \
                                            3*sum1n2//sum1n))
    n += 1
```

7.

```
def print_square(n):
    print(' ' + (n-2)*'-')
    for k in range(1, n-1):
        print('|' + (n-2)*' ' + '|')
    print(' ' + (n-2)*'-')
```

or

```
def print_square(n):
    print(' ' + (n-2)*'-' + '\n' + (n-2)*('|' + (n-2)*' ' + '|' + '\n')
                                                          + ' ' + (n-2)*'-')
```

8.

```
def my_pow(x, n):
    p = 1
    while n > 0:
        p *= x
        n -= 1
    return p
```

5 Strings, Lists, Dictionaries, and Files

Section 5.2

2.

```
def is_palindrome(word):
    return word == word[::-1]
```

Section 5.3

1.

```
def to_swedish(date):
    slash1 = date.find('/')
    slash2 = date.find('/', slash1+1)
    return (date[slash2+1:] + '-' +
            date[:slash1].rjust(2, '0') + '-' +
            date[slash1+1:slash2].rjust(2, '0'))
```

3.

```
def startswith(s, sub):
    return s[:len(sub)] == sub

def endswith(s, sub):
    if len(sub) == 0:
        return True
    return s[-len(sub):] == sub
```

4.

```
def get_digits(s):
    digits = ''
    for c in s:
        if c.isdigit():
            digits += c
    return digits
```

7.

```
def is_valid_amt(s):
    s = s.strip()
    dot = s.find('.')
    if dot == -1:
        return len(s) > 0 and s.isdigit()
    if dot == 0 or dot != len(s) - 3:
        return False
    return s[0:dot].isdigit() and s[dot+1:].isdigit()
```

8.

```
def remove_tag(s):
    start = s.find('<')
    if start == -1:
        return s
    end = s.find('>', start + 1)
    if end == -1:
        return s
    return s[:start] + s[end+1:]
```

Section 5.4

1.

```
def reverse(lst):
    i = 0
    j = len(lst) - 1
    while i < j:
        lst[i], lst[j] = lst[j], lst[i]
        i += 1
        j -= 1
```

4.

```
newnums = [x + 1 for x in nums]
```

7. The first time through the `for` loop, `x > a[i]` is executed twice; it is true once and false once. The next time through, the condition is true 3 times and false once, because `b[0]` has been inserted into `a`. The next time, it's true 5 times and false once. This pattern continues until the final time through the loop. Therefore, the total number of times the comparison is executed is

$$2+4+6+\ldots+196+198+199=\left(\frac{2+200}{2}\right)\cdot 100-1=10099\,.$$

```
def merge(a, b):
    i = 0  # Moved outside the while loop: no need to start
           #   searching from the beginning of a because b is sorted
    for x in b:
        while i < len(a) and x > a[i]:
            i += 1
        a.insert(i, x)
        i += 1  # Advance to the next position in a
```

This code executes `x > a[i]` $100\cdot 2-1=199$ times.

Section 5.5

2.

```
alligator = {'Title' : 'Alligator',
             'Band' : 'The Nationals',
             'Duration' : 4 * 60 + 5}
```

3.

```
def reverse_dictionary(d):
    new_d = {}
    for k in d.keys():
        new_d[d[k]] = k
    return new_d
```

Section 5.6

1.

```
pathname = input('File name: ').strip()
f = open(pathname)
line_number = 1
for line in f:
    print('{0:4d} {1:s}'.format(line_number, line), end='')
    count += 1
f.close()
```

3.

```
pathname = input('File name: ')
s = input('Target string: ')
f = open(pathname)
for line in f:
    if s in line:
        print(line, end='')
f.close()
```

6 Number Systems

Section 6.2

1. $10a+b=2(a+b) \Rightarrow 8a=b \Rightarrow a=1, b=8 \Rightarrow 18$ is the only two-digit number that is equal to twice the sum of its digits.

   $10a+b=7(a+b) \Rightarrow 3a=6b \Rightarrow a=2b \Rightarrow$ 21, 42, 63, 84 are the two-digit numbers that are equal to seven times the sum of their digits.

3. The number in each row has the form $100a+10b+c$. In the nine numbers in all rows, *a*, *b*, and *c* run through all values 1 through 9, so the sum of these numbers is $100s+10s+s$, where $s=1+2+3+4+5+6+7+8+9=45$. Therefore, the sum of the numbers in each column is $45(100+10+1)=45\cdot 111=4995$.

4. $15_{10}=9+2\cdot 3=120_3$
   $24_{10}=2\cdot 9+2\cdot 3=220_3$

6. $10011100_2=128+16+8+4=156_{10}$

9.

```
  201       201
+  12     -  12
  ---       ---
  220       112
```

10. The number is even if and only if the sum of its digits in base 3 is even, because

$$\left(a_n \cdot 3^n + a_{n-1} \cdot 3^{n-1} + ... + 3a_1 + a_0\right) - \left(a_n + a_{n-1} + ... + a_1 + a_0\right) =$$
$$\underbrace{222...2}_{n-1} \cdot a_n + \underbrace{22...2}_{n-2} \cdot a_{n-1} + ... + 2 \cdot a_1 = 2 \cdot \left(111...1 \cdot a_n + 11...1 \cdot a_{n-1} + ... + a_1\right)$$

So the sum of the digits of an even number must be divisible by 2, which means the number of 1s among the digits must be even.

Section 6.3

1. $1111111_2 = 2^7 - 1 = 127$ and $2^{15} - 1 = 32767$.

3. $010\,1110\,1011_2 = 2EB_{16}$

5. $4_{10} = 100_2$
$1011100_2 \cdot 4_{10} = 101110000_2$ (add two zeros on the right).
$1011100_2 / 4_{10} = 10111_2$ (remove two zeros on the right).

Section 6.4

1. $FFFFFACE_{16} = -532_{16} = -1330$

2. Because a `float` in Python has "only" 17 significant digits, the least significant digits in 1000000000.0000000001 are lost.

4.
```
>>> n = 1
>>> while int((17.0 / n) * n) == 17.0:
        n += 1
>>>
>>> n
4211
```

Section 6.5

1. There are $6 + 5 + 4 + 3 + 2 + 1 = 21$ pairs of integers (n, m) such that $0 < m < n \le 7$.

```
def pythagoreanTriple(m, n):
    return (n*n - m*m, 2*m*n, n*n + m*m)

for m in range(1, 7):
    for n in range(m+1, 8):
        print(pythagoreanTriple(m, n))
```

7 Boolean Algebra

Section 7.2

2. I did my homework and I went to see a movie.

3. Here or there

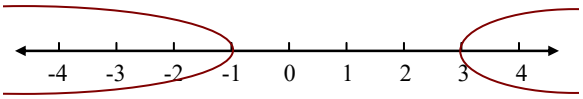
6.

P	*Q*	*R*	*Q or R*	***P* and (*Q* or *R*)**
F	F	F	F	**F**
F	F	T	T	**F**
F	T	F	T	**F**
F	T	T	T	**F**
T	F	F	F	**F**
T	F	T	T	**T**
T	T	F	T	**T**
T	T	T	T	**T**

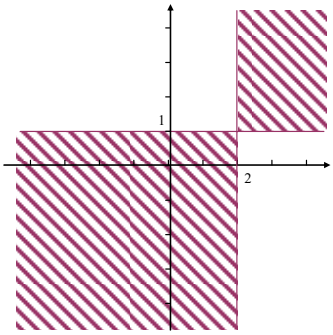
8. (*P* and (not *Q*)) or ((not *P*) and *Q*)

Section 7.3

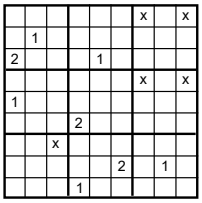
2.



4.



5.



6. 34% − 12% = 22% were only obese
14% − 12% = 2% had only diabetes
12% had diabetes and were obese
22% + 2% + 12% = 36% were obese, had diabetes, or were both obese and had diabetes.

Another way to look at it: $34+14-12=36$. This is an example of the *inclusion-exclusion principle*.

7. If A_p and A_q are the truth sets of p and q, respectively, then $p \to q$ if and only if $A_p \subseteq A_q$.

10. `s1 ^ s2 = (s1 | s2) - (s1 & s2)`

Section 7.4

1.
```
n > 0 and n % 2 == 0
```

3. Only (c)

4.
```
def num_days(month):
    if (month == 'January' or month == 'March' or month == 'May'
        month == 'July' or month == 'August' or month == 'October' or
        month == 'December'):
        return 31
    elif month == 'February':
        return 28
    elif (month == 'April' or month == 'June' or
          month == 'September' or month == 'November'):
        return 30
```

or

```
def num_days(month):
    if month in ('January', 'March', 'May', 'July', 'August'
                                          'October', 'December'):
        return 31
    elif month in ('April', 'June', 'September', 'November'):
        return 30
    elif month == 'February':
        return 28
```

5.
```
def is_leap_year(year):
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)
```

7.

```
('+' in s and '-' not in s) or ('-' in s and '+' not in s)
```

8.

```
def is_hex_digit(d):
    """ Return True if d is a single character and d
        is a hex digit: '0' - '9', 'a' - 'f' or 'A' - 'F';
        otherwise return False.
    """
    return len(d) == 1 and d in '0123456789abcdefABCDEF'
```

11.

```
def is_earlier_than(date1, date2):
    month1, day1, year1 = date1
    month2, day2, year2 = date2
    return (year1 < year2 or
            (year1 == year2 and month1 < month2) or
            (year1 == year2 and month1 == month2 and day1 < day2))
```

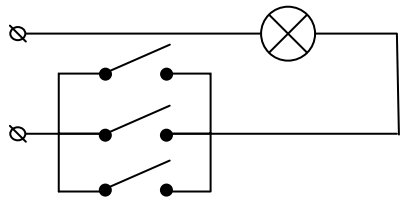
Another solution:

```
def is_earlier(date1, date2):
    month1, day1, year1 = date1
    month2, day2, year2 = date2
    if year1 < year2:
        return True
    elif year1 > year2:
        return False
    if month1 < month2:
        return True
    elif month1 > month2:
        return False
    return day1 < day2
```

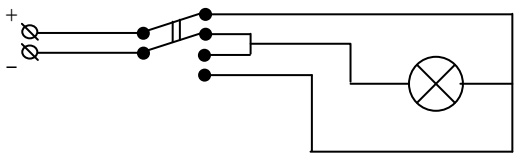
8 Digital Circuits and Bitwise Operators

Section 8.2

1.

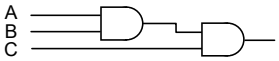


3.

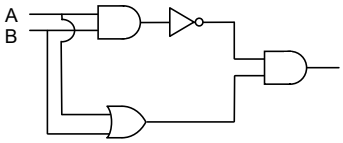


5. AND

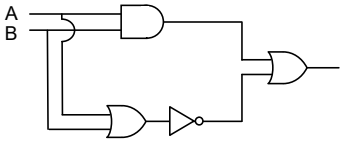
7.



8.



9.



Section 8.3

1.
```
a & b = 01000110
a | b = 11110111
a ^ b = 10110001
a << 1 = 11001110
b >> 2 = 00110101
```

3.
```
if (status_reg & 0x0008) != 0 and (status_reg & 0x0020) != 0
    ...
```

4.
```
def divisible_by8(n):
    """Return True if n is divisible by 8;
       otherwise return false.
    """
    return n & 0x07 == 0
```

5.
```
pattern ^= 0x00ffffff
```

7.
```
byte & 0x2A != 0
```

9.

```
def rev_bits(n):
    x = n & 0xC0000000
    bit1 = 1
    bit2 = 0x20000000
    for k in range(29, 0, -2):
        x |= (n & bit1) << k
        x |= (n & bit2) >> k
        bit1 <<= 1
        bit2 >>= 1
    return x
```

12. The `&` and `|` operators can be applied to Boolean expressions, but they do not use short-circuit evaluation. Try, for example,

```
>>> x = 0
>>> x != 0 & 1/x > 1
ZeroDivisionError: division by zero
```

9 Turtle Graphics

Section 9.2

See `PY\PythonCode\`**`TurtleExercises.py`**.

Section 9.3

See `PY\PythonCode\`**`TurtleExercises.py`**.

Section 9.4

1. $2^{24} = 16{,}777{,}216$

3. 1920/1200 = 1.6; the golden ratio is approximately 1.618. Pretty close.

5 See `PY\PythonCode\`**`TurtleExercises.py`**.

10 Sequences and Sums

Section 10.2

2. $a_n = \dfrac{1}{n(n+1)}$

4. $a_7 - a_1 = 6d = 21 - 3 = 18 \Rightarrow d = 3 \Rightarrow a_{12} = a_1 + 11d = 36$

7. $a_n = a_{n-1} + d$; $a_{n+1} = a_n + d \Rightarrow a_n = a_{n+1} - d$. Adding these two equalities together, we get $2a_n = a_{n-1} + a_{n+1} \Rightarrow a_n = \frac{a_{n-1} + a_{n+1}}{2}$.

9. Yes; for example, 0, 0, 0, ... or 1, 1, 1, ... or any other sequence in which all terms have the same value.

10. 1

Section 10.3

3. The sum of the first *n* odd numbers is $\sum_{k=1}^{n}(2k-1) = n\frac{1+(2n-1)}{2} = n\frac{2n}{2} = n^2$.

6. $s_n = a + ar + ar^2 + ... + ar^{n-1}$

$rs_n = ar + ar^2 + ar^3 + ... + ar^n \Rightarrow rs_n - s_n = ar^n - a \Rightarrow s_n = a\frac{r^n - 1}{r-1}$.

If $a = \frac{1}{2}$ and $r = \frac{1}{2}$, then $s_n = \frac{1}{2} \cdot \frac{1 - \frac{1}{2^n}}{1 - \frac{1}{2}} = \frac{1}{2} \cdot \frac{1 - \frac{1}{2^n}}{\frac{1}{2}} = 1 - \frac{1}{2^n}$.

7. $\sum_{d=1}^{6} d \cdot 10^d = 6543210$

9. $k(k+1)(k+2) - (k-1)k(k+1) = 3k(k+1) = 3k^2 + 3k$, so $k^2 = \frac{k(k+1)(k+2) - (k-1)k(k+1)}{3} - k \Rightarrow$

$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(n+2)}{3} - \frac{n(n+1)}{2} = \frac{n(n+1)\big(2(n+2)-3\big)}{6} = \frac{n(n+1)(2n+1)}{6}$.

Section 10.4

2. $\sum_{n=1}^{\infty} \frac{n+1}{100n}$ diverges because $\frac{n+1}{100n} > \frac{1}{100}$.

4. $\sum_{k=0}^{n} r^k = \frac{1 - r^{n+1}}{1-r}$. The series converges to $\frac{1}{1-r}$ if and only if $|r| < 1$, that is, $-1 < r < 1$.

5. $1 + \frac{1}{3} + \frac{1}{5} + ... + \frac{1}{2n-1} + ...$ diverges. Indeed,

$1 + \frac{1}{3} + \frac{1}{5} + ... + \frac{1}{2n-1} + ... > \frac{1}{2} + \frac{1}{4} + \frac{1}{6} + ... + \frac{1}{2n} + ... = \frac{1}{2}\left(1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} + ...\right)$, and the harmonic series diverges.

7. $\sum_{n=1}^{\infty}\frac{1}{n^3}$ converges because $0<\frac{1}{n^3}\le\frac{1}{n^2}<\frac{1}{n(n+1)}$, and we know that the telescopic series $\sum\frac{1}{n(n+1)}$ converges.

8. Let P_n be the perimeter of the snowflake after *n* iterations. Then $P_n=\frac{4}{3}P_{n-1}=\left(\frac{4}{3}\right)^n P$, so the perimeter of the snowflake increases without bound.

On the first iteration we add $3\cdot\frac{1}{9}A=\frac{A}{3}$ to the area. On the second iteration we add $3\cdot4\cdot\frac{1}{81}A=\frac{4}{9}\cdot\frac{A}{3}$. On the third iteration we add $3\cdot4^2\frac{1}{9^3}A=\left(\frac{4}{9}\right)^2\cdot\frac{A}{3}$. And so on. So, after *n* iterations, the area is $A+\frac{A}{3}\left[1+\frac{4}{9}+\left(\frac{4}{9}\right)^2+\ldots+\left(\frac{4}{9}\right)^{n-1}\right]$. As we know, the geometric series $1+\frac{4}{9}+\left(\frac{4}{9}\right)^2+\ldots+\left(\frac{4}{9}\right)^{n-1}+\ldots$ converges to $\frac{1}{1-\frac{4}{9}}=\frac{9}{5}$. So, the area of the snowflake converges to $A+\frac{A}{3}\cdot\frac{9}{5}=\frac{8}{5}A$. Thus the Koch Snowflake has an "infinite" perimeter and a finite area!

Section 10.5

1. $F_1=1$ is odd; $F_2=1$ is odd; $F_3=2$ is even. The Fibonacci sequence goes odd, odd, even, odd, odd, even, ... because the sum of two odd numbers is even and the sum of an even and an odd number is odd. Therefore, every third Fibonacci number is even. F_{999} is even; F_{1000} is odd.

4. There is one path from *A* and one path from *B* to C_1. There is one path from *A* and there are two paths from *B* to C_2. There are two paths from *A* and three paths from *B* to C_3. Let $P_A(n)$ and $P_B(n)$ be the number of paths from *A* to C_n and from *B* to C_n, respectively. It looks like $P_A(n)=F_n$ and $P_B(n)=F_{n+1}$. Indeed, any path from *A* or from *B* to C_n must go through either C_{n-1} or C_{n-2}, so $P_A(n)=P_A(n-1)+P_A(n-2)$ and $P_B(n)=P_B(n-1)+P_B(n-2)$ — Fibonacci numbers again!

5. (a)

```
def fibonacci_list(n):
    """ Return a list fibs of length n+1 in which fibs[0] is 0
        and fibs[k] is the k-th Fibonacci number for 1 <= k <= n.
    """
    fibs = (n+1)*[0]
    fibs[1] = fibs[2] = 1
    for k in range(3, n+1):
        fibs[k] = fibs[k-1] + fibs[k-2]
    return fibs
```

(c)

```
def fibonacci(n):
    """ Return the n-th Fibonacci number, where F(1) = F(2) = 1."""
    f1 = f2 = 1
    for k in range(3, n+1):
        f1, f2 = f2, f1 + f2
    return f2
```

This code is also included in `Py\PythonCode\`**`Fibonacci.py`**.

6.

```
f = fibonacci_list(10)
s = 0
for n in range(1, 10, 2):
    s += f[n]
    print(n, s)
```

The output is:

```
1 1
3 3
5 8
7 21
9 55
```

This code is also included in `Py\PythonCode\`**`Fibonacci.py`**.

It looks like $F_1 + F_3 + \ldots + F_{2n-1} = F_{2n}$. Indeed, this is true for n = 1, 2, 3, 4, 5. Suppose it is true for n. Then $F_1 + F_3 + \ldots + F_{2n-1} + F_{2n+1} = F_{2n} + F_{2n+1} = F_{2n+2} = F_{2(n+1)}$ — so it is also true for n+1. It must be true for all n.

7. $F_n = \dfrac{1}{\sqrt{5}}\left(\varphi^n - \overline{\varphi}^n\right) \;\Rightarrow\; \left|F_n - \dfrac{\varphi^n}{\sqrt{5}}\right| = \left|\dfrac{\overline{\varphi}^n}{\sqrt{5}}\right| < \dfrac{1}{2}.$

9. $F_{2n+1} = F_n^2 + F_{n+1}^2$ and $F_{2n-1} = F_{n-1}^2 + F_n^2 \;\Rightarrow$
$F_{2n} = F_{2n+1} - F_{2n-1} = F_{n+1}^2 - F_{n-1}^2 = (F_n + F_{n-1})^2 - F_{n-1}^2 = F_n^2 + 2F_nF_{n-1}$, Q.E.D.

10.

```
x = 1
for n in range(1, 101):
    print(n, x)
    x = 1 + 1/x
```

This sequence converges to the golden ratio.

11 Parity, Invariants, and Finite Strategy Games

Section 11.2

2. $2^7 = 128$. The first seven bits can be anything; the eighth bit is determined by the first seven.

4. The 0 bit in the third row and fourth column (because the parities of this row and this column are odd).

5.

```
def correct_error(t):
    """Correct a parity error in one bit in a rectangular table."""
    n_rows, n_cols = len(t), len(t[0])
    error_row = error_col = -1

    # Find the index of the row with error:
    for r in range(n_rows):
        if t[r].count('1') % 2 == 1:
            error_row = r

    if error_row == -1:
        return

    # Find the index of the column with error:
    for c in range(n_cols):
        count = 0
        for r in range(n_rows):
            count += int(t[r][c])
        if count % 2 != 0:
            error_col = c

    # Correct the error:
    s = t[error_row]
    d = '0' if s[error_col] == '1' else '1'
    t[error_row] = s[0:error_col] + d + s[error_col+1:]

t = ['011011', '100010', '101001', '010100']
print(t)
correct_error(t)
print(t)
```

6. The last column and the last row of the table are determined by the first three rows and the first five columns. The number of tables with even parity is equal to the number of tables with odd parity — both numbers are equal to 2^{15}.

7.

```
def is_valid_UPC(s):
    """ Return True if s represents a valid UPC string of 12 digits."""
    c_sum = 3*sum(int(d) for d in s[0::2]) + sum(int(d) for d in s[1::2])
    return c_sum % 10 == 0

print(is_valid_UPC('072043000187'))
print(is_valid_UPC('072043000188'))
print(is_valid_UPC('072040300187'))
```

This code is also included in `Py\PythonCode\`**`Checksums.py`**.

8. (a) It does. The numbers $3 \cdot d$, for d from 0 to 9, all end with different digits, so if you change one digit in the UPC, the checksum will no longer be 0.

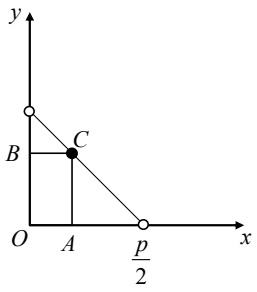
(b) If we were to multiply by 2, then 5 could be substituted for 0, and vice-versa, and the checksum would remain unchanged.

(c) It does not. We can transpose any 2 digits whose difference is 5, such as 1 and 6. $3 \cdot 1 + 6 = 3 \cdot 6 + 1 \mod 10$.

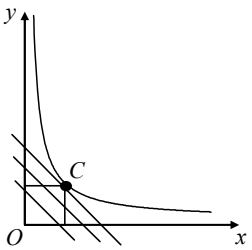
Section 11.3

1. The difference between the number of uncovered black squares and the number of uncovered white squares remains constant, because each domino always covers one black square and one white square. At the beginning the difference is 2, so it can't be 0 at the end.

3. It is an open interval on the diagonal line $x + y = \frac{p}{2}$:



5. As we have seen in question 4, the area of the rectangle is constant when its vertex *C* lies on the hyperbola, such as $y = \frac{1}{x}$. The perimeter is constant when *C* is on a diagonal line $x + y = \frac{p}{2}$. For the smallest perimeter we need to find the diagonal line that is closest to the origin, yet intersects with, or at least touches, the hyperbola. As we shift the diagonal line parallel to itself away from the origin, it eventually touches the hyperbola at (1, 1). That point gives the rectangle with the smallest perimeter; that rectangle is a square.



6. The result does not depend at all on the order of pairs chosen. To see this, replace each plus with +1 and each minus with −1. Then the described operation is equivalent to replacing a pair of numbers with their product. The end result is the product of the original numbers.

9. For example, in

```
def my_pow(x, n):
    k = 0
    p = 1
    while k < n:
        p *= x
        k += 1
    return p
```

the loop invariant is $p = x^k$.

Section 11.4

1. The number of stones remaining in the pile is evenly divisible by 5.

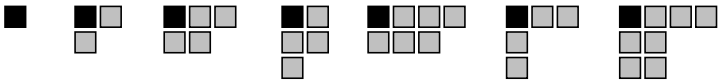
3.

−	+	−	+	−	+	−	+
−	−	−	−	−	−	−	−
−	+	−	+	−	+	−	+
−	−	−	−	−	−	−	−
−	+	−	+	−	+	−	+
−	−	−	−	−	−	−	−
−	+	−	+	−	+	−	+
−	−	−	−	−	−	−	−

The first player can win by moving to a “plus” square on the first move.

4. Initially there are two piles of stones with equal numbers of stones in them. A player must either take one stone from either pile, or take two stones, one from each pile. The safe positions are the ones in which the number of remaining stones in each pile is even.

7.



The correct first move is to go to the last position shown above.

8.

```
3:    011
4:    100
5:    101
6:    110
```

The number of 1s in the leftmost column is odd, so this is not a safe position.

9.

```
6:    0110
8:    1000
11:   1011
```

We need to flip the bits in the second and fourth columns in the '6' pile:

```
3:    0011
8:    1000
11:   1011
```

In other words, we need to take three stones from the '6' pile.

12. Let's call every other stack "white" and the rest "black." Suppose you go first. Note that before your first move and before each of your moves, the colors of the stacks on the two ends are different. After your move these colors are the same, and your opponent is forced to take one of them. At the outset, count the total number of coins in the white stacks and compare it to the total number of coins in the black stacks. Pick the color that has more coins in it and stick with this color: always take the stack of that color, forcing your opponent to take a stack of the opposite color. This strategy works for any even number of stacks.

12 Counting

Section 12.2

1. $10 \cdot 10 \cdot 10 = 1000$

3. $21 \cdot 5 \cdot 21 = 2205$

7. $2^8 \cdot 2^8 \cdot 2^8 = 2^{24} = 16777216$

8. $3^5 = 243$. First choose a peg for the largest disk, then for the second largest, and so on.

Section 12.3

1. $9 \cdot 9 \cdot 8 = 648$

4. $8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 20160$

5. The same answer as in Question 4, only this time you "seat" chairs on people, not people on chairs.

7. $9 \cdot 8 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot \frac{1}{2} = 725760$ mins $= 12096$ hours = 504 days at 24 hours/day.
   $9 \cdot 8$ for M, S; $\frac{1}{2}$ for solving, on average, after half of the trials.

8. BINARY: $\frac{6!}{360} = 2$ seconds
   DECIMAL: $\frac{7!}{360} = 14$ seconds
   CONVERSATION: $\frac{12!}{360} = 1330560$ seconds = 369.6 hours = 15.4 days

Section 12.4

1. $\frac{12 \cdot 11}{2} = 66$

4. $\frac{4!}{2} = 12$. Seat the hosts first; 4! ways to seat the guests, divided by 2 for mirror arrangements.

5. $\frac{6!}{6 \cdot 4} = 30$. 6! ways to color a cube in a fixed position divided by 24 ways to position the cube: 6 ways to stand the cube on one side and 4 ways to rotate the cube on that side.

6. $6 \cdot 12$ for the first flight; $5 \cdot 12$ for the second flight; $4 \cdot 12$ for the third flight. But the order of flights does not matter, so we have to divide the previous result by the number of permutations of 3. The answer is $6 \cdot 12 \cdot 5 \cdot 12 \cdot 4 \cdot 12/3! = 34560$.

7. $8! = 40320$. There must be a rook in each row; 8 ways to choose a rook's position in the first row, times 7 ways to choose a rook's position in the second row, and so on.

Section 12.5

1. 10

3. $\binom{40}{5} = \binom{40}{35} = 658008$

5. $\binom{9}{3} \cdot \binom{6}{3} = 1680$

7. 13 ways to choose the first rank; 4 ways to choose 3 cards of that rank; 12 ways to choose the second rank; 6 ways to choose 2 cards of that rank. $13 \cdot 4 \cdot 12 \cdot 6 = 3744$.

9. The number of different ways to reshuffle is $\binom{6}{3} = 20$. The number of different truth tables is $2^4 = 16$. So not all arrangements result in different truth tables.

Section 12.6

1. $12 + 6 + 2 = 20$

3. 34

n	m
2	none
3	2
4	3
5	2, 3, 4
6	5
7	2, 3, 4, 5, 6
8	3, 5, 7
9	2, 4, 5, 7, 8
10	3, 7, 9
11	2, 3, 4, 5, 6, 7, 8, 9, 10
12	5, 7, 11

6. $3a$: (7+7+7) $\Rightarrow \binom{4}{3} = 4$

$2a+b$:6+6+9; 8+8+5; 9+9+3 $\Rightarrow \binom{4}{2}\cdot\binom{4}{1}\cdot 3 = 72$

$a + b + c$: 2+ 9+10; 3+8+10; 4+7+10; 4+8+9; 5+6+10; 5+7+9; 6+7+8 $\Rightarrow \binom{4}{1}\cdot\binom{4}{1}\cdot\binom{4}{1}\cdot 7 = 448$

$4 + 72 + 448 = 524$

7. $9\cdot 10^3 - 8\cdot 9^3 = 3168$

9. The total number of paths is $\binom{6}{3} = 20$. The number of paths that go through C is $\binom{4}{2}\cdot\binom{2}{1} = 12$. The number of paths that do not go through C is $20 - 12 = 8$.

10. $62^5 - 36^5 - 52^5 + 26^5 + 62^4 - 36^4 - 52^4 + 26^4 = 493586080$

13 Probabilities

Section 13.2

1. $\frac{1}{5}$

3. $\frac{1}{\binom{36}{6}} = \frac{1}{1947792}$

5. $\frac{1}{38}$ and $\frac{18}{38}$

Section 13.3

1. $\frac{13\cdot 48}{\binom{52}{5}} = \frac{1}{4165}$

2. $\frac{4}{\binom{52}{5}} = \frac{1}{649740}$

3. $\dfrac{\binom{70}{10}}{\binom{80}{20}} \approx 1.122 \times 10^{-7}$

5. $\dfrac{52 \cdot 51 \cdot 50 - 52 \cdot 48 \cdot 44}{52 \cdot 51 \cdot 50} = \dfrac{73}{425}$

7. $\dfrac{\text{count}(0) + \text{count}(1) + \text{count}(2)}{\text{total \# of arrangements}} = \dfrac{5^{16} + \binom{16}{1} \cdot 5^{15} + \binom{16}{2} \cdot 5^{14}}{6^{16}} \approx 0.48679$

8. All different ranks — 10 possibilities:
2+8+11; 2+9+10; 3+7+11; 3+8+10; 4+6+11; 4+7+10; 4+8+9; 5+6+10; 5+7+9; 6+7+8
Two of the same rank, different third — 6 possibilities:
5+5+11; 6+6+9; 8+8+5; 9+9+3; 10+10+1; 1+11+9
All three the same — 1 possibility: 7+7+7
This results in
$$\frac{10 \cdot \binom{4}{1} \cdot \binom{4}{1} \cdot \binom{4}{1} + 6 \cdot \binom{4}{2}\binom{4}{1} + 1 \cdot \binom{4}{3}}{\binom{52}{3}} = \frac{788}{22100}$$

Section 13.4

1. $\dfrac{1}{6} \cdot \dfrac{1}{6} \cdot \dfrac{1}{6} = \dfrac{1}{216}$

3. $\dfrac{5}{6} \cdot \dfrac{5}{6} \cdot \dfrac{5}{6} = \dfrac{125}{216}$ and $1 - \dfrac{125}{216} = \dfrac{91}{216}$

5. $\dfrac{3}{10} \cdot \dfrac{4}{10} = \dfrac{3}{25}$

6. Win after 1 rally: $\dfrac{2}{3}$

Win after 1 or 3 rallies: $\dfrac{2}{3} + \dfrac{1}{3} \cdot \dfrac{2}{3} \cdot \dfrac{2}{3} = \dfrac{2}{3}\left(1 + \dfrac{2}{9}\right) = \dfrac{22}{27}$

Win after 1, 3, or 5 rallies: $\dfrac{2}{3} + \dfrac{1}{3} \cdot \dfrac{2}{3} \cdot \dfrac{2}{3} + \dfrac{1}{3} \cdot \dfrac{2}{3} \cdot \dfrac{1}{3} \cdot \dfrac{2}{3} \cdot \dfrac{2}{3} = \dfrac{2}{3}\left(1 + \dfrac{2}{9} + \left(\dfrac{2}{9}\right)^2\right) = \dfrac{206}{243}$

Win eventually: $\dfrac{2}{3}\left(1 + \dfrac{2}{9} + \left(\dfrac{2}{9}\right)^2 + \ldots\right) = \dfrac{2}{3} \cdot \dfrac{1}{1 - \dfrac{2}{9}} = \dfrac{6}{7}$

8. $\left(\frac{1}{30}\right)^2+\left(\frac{5}{30}\right)^2+\left(\frac{20}{30}\right)^2=\frac{426}{900}$

Section 13.5

1.

```
from random import choice, shuffle, sample

def random_hand():
    """Generate a random hand of 5 cards."""
    deck = [(s, r) for s in 'SHDC' for r in range(1, 14)]
    hand = [];
    while len(hand) < 5:
        card = choice(deck)
        hand.append(card)
        deck.remove(card)
    return hand
```

Or, shorter:

```
def random_hand():
    """Generate a random hand of 5 cards."""
    deck = [(s, r) for s in 'SHDC' for r in range(1, 14)]
    shuffle(deck);
    return deck[:5]
```

Or, even shorter:

```
def random_hand():
    """Generate a random hand of 5 cards."""
    deck = [(s, r) for s in 'SHDC' for r in range(1, 14)]
    return sample(deck, 5)
```

2. $\left(\frac{5}{26}\right)^3=\frac{125}{17576}$

8.

```
from random import random

def positive_choice(s):
    """Return a positive integer randomly chosen from s."""
    k = 0
    r = None
    for x in s:
        if x > 0:
            k += 1
            if random() < 1 / k:
                r = x
    return r

s = [-1, 1, 2, 0, 3, -2, 4, 5, -6]
counts = (max(s) + 1)*[0]
n = 10000
for t in range(n):
    counts[positive_choice(s)] += 1
print(counts)
```

This code is also included in `Py\PythonCode\`**`RandomQuestions.py`**.

14 Vectors and Matrices

Section 14.2

1.

$\vec{u} = (4, 3)$ $|\vec{u}| = \sqrt{4^2 + 3^2} = \sqrt{25} = 5$

$\vec{v} = (2, 2)$ $|\vec{v}| = \sqrt{2^2 + 2^2} = \sqrt{8} = 2\sqrt{2}$

$\vec{w} = (-2, 1)$ $|\vec{w}| = \sqrt{(-2)^2 + 1^2} = \sqrt{5}$

3. (a) $\sqrt{5^2 + 12^2} = \sqrt{169} = 13$ (b) $\sqrt{1^2 + 7^2} = \sqrt{50} = 5\sqrt{2}$ (c) $\sqrt{(-3)^2 + 4^2} = \sqrt{25} = 5$
(d) $\sqrt{3^2 + 4^2 + 12^2} = \sqrt{169} = 13$

4. $\vec{u} = (-3, 4)$; $|\vec{u}| = 5 \Rightarrow$ unit vector with the same direction is $\dfrac{\vec{u}}{|\vec{u}|} = \left(-\dfrac{3}{5}, \dfrac{4}{5}\right)$.

5. (a) $\vec{u} = (2, 2)$ $\vec{v} = (2, -2)$ $\vec{u} + \vec{v} = (4, 0)$ $\vec{u} - \vec{v} = (0, 4)$
(b) $\vec{u} = (2, 2)$ $\vec{v} = (2, 1)$ $\vec{u} + \vec{v} = (4, 3)$ $\vec{u} - \vec{v} = (0, 1)$
(c) $\vec{u} = (0, 2)$ $\vec{v} = (-3, -2)$ $\vec{u} + \vec{v} = (-3, 0)$ $\vec{u} - \vec{v} = (3, 4)$
(d) $\vec{u} = (2, 0)$ $\vec{v} = (3, 0)$ $\vec{u} + \vec{v} = (5, 0)$ $\vec{u} - \vec{v} = (-1, 0)$
(e) $\vec{u} = (0, 2)$ $\vec{v} = (0, -3)$ $\vec{u} + \vec{v} = (0, -1)$ $\vec{u} - \vec{v} = (0, 5)$
(f) $\vec{u} = (0, 2)$ $\vec{v} = (0, -2)$ $\vec{u} + \vec{v} = (0, 0)$ $\vec{u} - \vec{v} = (0, 4)$

7. Choose the point $(1, 2)$ on the line $y = 2x$ and the point $(2, 1)$ on the line $y = \dfrac{x}{2}$. If $\vec{u} = (1, 2)$, $\vec{v} = (2, 1)$, then $\vec{u} \cdot \vec{v} = 4$, $|\vec{u}| = |\vec{v}| = \sqrt{5} \Rightarrow \cos\theta = \dfrac{4}{5} \Rightarrow \theta \approx 0.6435$ radians.

8. $(-5, 1)$ and $(5, -1)$

13. Take $\vec{u} = (x, y, z)$ and $\vec{v} = (1, 1, 1)$. Then $\vec{u} \cdot \vec{v} = x + y + z = 1$, $|\vec{u}|^2 = x^2 + y^2 + z^2$ and $|\vec{v}|^2 = 3$.
$(\vec{u} \cdot \vec{v})^2 \le |\vec{u}|^2 \cdot |\vec{v}|^2 \Rightarrow 1 \le 3(x^2 + y^2 + z^2) \Rightarrow x^2 + y^2 + z^2 \ge \dfrac{1}{3}$, Q.E.D.

Section 14.3

2.

```
def display_matrix(m):
    """Print out m with aligned columns."""
    for row in m:
        for x in row:
            print('{0:4d}'.format(x), end = '')
        print()
    print()

m = [[1, 2, 3],
     [11, 21, 31],
     [21, 22, 33]]
display_matrix(m)
```

Output:

```
   1   2   3
  11  21  31
  21  22  33
```

This code is also included in `Py\PythonCode\`**`VectorsAndMatrices.py`**.

5.

```
def matrix_dot_vector(A, u):
    """Return the vector Av."""
    n = len(u)
    return [sum(A[r][c]*u[c] for c in range(n)) for r in range(n)]
```

or

```
def matrix_dot_vector(A, u):
    """Return the vector Av."""
    return [dot_product(row, u) for row in A]


A = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

u = (100, 10, 1)
v = matrix_dot_vector(A, u)
print(v)
```

Output:

```
[123, 456, 789]
```

This code is also included in `Py\PythonCode\`**`VectorsAndMatrices.py`**.

8.

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{pmatrix}$$

$$\left(x\cos\theta - y\sin\theta\right)^2 + \left(x\sin\theta + y\cos\theta\right)^2 =$$
$$x^2\left(\cos\theta\right)^2 - 2xy\sin\theta\cos\theta + y^2\left(\sin\theta\right)^2 +$$
$$x^2\left(\sin\theta\right)^2 + 2xy\sin\theta\cos\theta + y^2\left(\cos\theta\right)^2 =$$
$$x^2\left(\left(\sin\theta\right)^2 + \left(\cos\theta\right)^2\right)^2 + y^2\left(\left(\sin\theta\right)^2 + \left(\cos\theta\right)^2\right) = x^2 + y^2$$

Q.E.D.

11.

```
def determinant3(a):
    return (a[0][0]*a[1][1]*a[2][2] +
            a[0][1]*a[1][2]*a[2][0] +
            a[1][0]*a[2][1]*a[0][2] -
            a[0][2]*a[1][1]*a[2][0] -
            a[0][1]*a[1][0]*a[2][2] -
            a[2][1]*a[1][2]*a[0][0])

def copy_matrix(a):
    """Return a copy of a."""
    return [a[0][:], a[1][:], a[2][:]]

def solve3(a, c):
    """ Solve the system of three linear equations ax = c."""
    solution = 3*[None]
    d = determinant3(a)
    if d == 0:
        return solution
    for i in range(3):
        b = copy_matrix(a)
        b[0][i] = c[0]
        b[1][i] = c[1]
        b[2][i] = c[2]
        solution[i] = determinant3(b) / d
    return solution

a = [[2, 3, 1],
     [4, 6, 5],
     [7, 9, 8]]

c = [42, 96, 150]
print('determinant = ', determinant3(a))
print(solve3(a, c))
```

Output:

```
determinant =  9
[4.0, 10.0, 4.0]
```

This code is also included in `Py\PythonCode\`**`VectorsAndMatrices.py`**.

15 Polynomials

Section 15.2

1. $\left(x^4+3x^2+1\right)+\left(x^3+x^2+8\right)=x^4+x^3+4x^2+9$

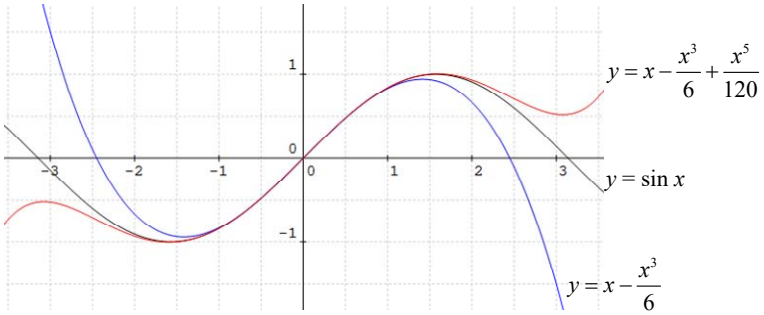
4.
```
def negate(p):
    """Return -p for a polynomial p."""
    return [-a for a in p]

p = [1 ,-2, 3]
print(negate(p))
```

Output:

```
[-1, 2, -3]
```

This code is also included in `Py\PythonCode\`**`Polynomials.py`**.

8.



$x-\frac{x^3}{6}$ approximates $\sin x$ quite well for $-1 \le x \le 1$; $x-\frac{x^3}{6}+\frac{x^5}{120}$ approximates $\sin x$ very well for $-2 \le x \le 2$.

Section 15.3

1.
```
def reduce(p):
    """Divide p by the leading coefficient."""
    f = 1/p[0]
    result = [f*a for a in p]
    result[0] = 1
    return result
```

This code is also included in `Py\PythonCode\`**`Polynomials.py`**.

4.

```
def multiply(p1, p2):
    """Return p1*p2."""
    n = len(p1) + len(p2) - 1
    result = n*[0]
    for i in range(len(p1)):
        for j in range(len(p2)):
            result[i+j] += p1[i] * p2[j]
    return result
```

This code is also included in `Py\PythonCode\`**`Polynomials.py`**.

7.

```
n = int(input('Enter a positive integer: '))
p = [1]  # 1
x_plus_1 = [1, 1]  # x + 1

while n > 0:
    p = multiply(p, x_plus_1)
    n -= 1

print(p)
print(sum(p))
```

For instance, if the user enters `4`, the output is `[1, 4, 6, 4, 1] 16`.

This code is also included in `Py\PythonCode\`**`Polynomials.py`**.

10. $(x-u)(x-v) = x^2 - (u+v)x + uv = x^2 - px + q \;\Rightarrow u, v = \dfrac{p \pm \sqrt{p^2 - 4q}}{2}$ and $\dfrac{p - \sqrt{p^2 - 4q}}{2}$.

Section 15.4

2. For each element of the set, there are two possibilities: it is either in the subset or not. Therefore, the number of possible subsets of a set of *n* elements is 2^n. $\binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{n} = 2^n$.

4. $\displaystyle\sum_{k=0}^{n} 2^k \binom{n}{k} = (1+2)^n = 3^n$

7.

```
def pascal_triangle(n):
    """ Return the first n rows of Pascal's triangle in a list."""
    t = [[1]]
    row = [1]
    for k in range(1, n):
        row = [1] + [row[i] + row[i+1] for i in range(k-1)] + [1]
        t.append(row)
    return t

n = int(input('Enter a positive integer: '))
t = pascal_triangle(n)
for row in t:
    print(row)
```

This code is also included in `Py\PythonCode\`**`PascalTriangle.py`**.

8. When $n = 1$, $\binom{2}{1} = \binom{2}{0} + \binom{2}{2}$. When $n > 1$,

$$\binom{2n}{n-1} = \binom{2n-1}{n-2} + \binom{2n-1}{n-1}$$

$$\binom{2n}{n+1} = \binom{2n-1}{n} + \binom{2n-1}{n+1} \Rightarrow$$

$$\binom{2n}{n-1} + \binom{2n}{n+1} = \binom{2n-1}{n-2} + \binom{2n-1}{n-1} + \binom{2n-1}{n} + \binom{2n-1}{n+1} \geq \binom{2n-1}{n-1} + \binom{2n-1}{n} = \binom{2n}{n} \Rightarrow$$

$\binom{2n}{n} \leq \binom{2n}{n-1} + \binom{2n}{n+1}$, Q.E.D.

16 Recurrence Relations and Recursion

Section 16.2

1. 15

3. $f(n) = \begin{cases} 2, \text{ if } n = 1 \\ 2 \cdot f(n-1), \text{ if } n > 1 \end{cases}$

4. $f(n) = \begin{cases} 10, \text{ if } n = 1 \\ f(n-1) + 20, \text{ if } n > 1 \end{cases}$

7. $f(n) = \begin{cases} 6, \text{ if } n = 3 \\ \frac{n}{n-3} \cdot f(n-1), \text{ if } n > 3 \end{cases}$

Section 16.3

1.

```
def factorial(n):
    """Return n-factorial for n >= 0."""
    if n == 0:  # 0! is 1
        return 1
    return n * factorial(n - 1)
```

This code is also included in `Py\PythonCode\`**`Recursion.py`**.

3.

```
def eval_polynomial(p, x):
    """Return p(x)."""
    if len(p) == 1:
        return(p[0])
    return x*eval_polynomial(p[:-1], x) + p[-1]

print(eval_polynomial([1, 2, 3], 2))
```

This code is also included in `Py\PythonCode\`**`Recursion.py`**.

4. (a)

```
def print_digits(n):
    """Print a triangle made of digits."""
    print(n * str(n))
    if n > 1:
        print_digits(n - 1)
```

(b)

```
def print_digits(n):
    """Print an inverted triangle made of digits."""
    if n > 1:
        printDigits(n - 1)
    print(n * str(n))
```

5. If the three pegs are numbered 1, 2, and 3, the sum of the numbers is 6, so knowing two pegs you can find the number of the third by subtracting the total of the two known pegs from 6. Therefore, `spare_peg = 6 - from+peg - to_peg` makes `spare_peg` equal to the number not used as `from_peg` or `to_peg`.

6. The implicit base case is `n == 1`, when the program just moves one disk from `from_peg` to `to_peg`.

8.

2 disks: 3 moves
3 disks: 7 moves
4 disks: 15 moves
n disks: $2^n - 1$ moves

The "lifespan of the universe" is about 584,942,417,355 years.

Section 16.4

1. Base Case: $f_1 = 1 = 1^2$.
   Induction Step: Suppose $f_n = n^2$. Then $f_{n+1} = f_n + 2n + 1 = n^2 + 2n + 1 = (n+1)^2$.

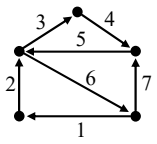
3. $L_n = 2L_{n-1} + (n-2)$
   Base Case: $L_2 = 2^{2-1} - 2 = 0$.
   Induction Step: $L_n = 2L_{n-1} + (n-2) = 2[2^{n-2} - (n-1)] + n - 2 = 2^{n-1} - 2n + 2 + n - 2 = 2^{n-1} - n$.

4. Let R_n be the number of regions into which *n* lines divide the plane. We will prove, using math induction, that $R_n = \frac{n(n+1)}{2} + 1$. This is true for $n = 1$, because $R_1 = \frac{1 \cdot 2}{2} + 1 = 2$. Suppose this is true for any number of lines $k < n$ (induction hypothesis). In particular, $R_{n-1} = \frac{(n-1)n}{2} + 1$. When we add the *n*-th line, it is cut into *n* segments by the existing lines; each of these segments cuts an existing region into two, adding one region. Therefore, $R_n = R_{n-1} + n$. Using this recurrence relation and the induction hypothesis, we get $R_n = \left[\frac{(n-1)n}{2} + 1\right] + n = \frac{n(n+1)}{2} + 1$.

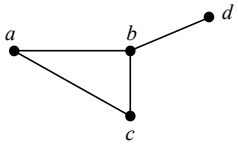
17 Graphs

Section 17.2

1. For example:



2.



4. (a) and (b) are simple graphs; (c) is a multigraph; (d) has a loop

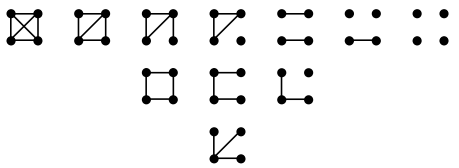
5. (b) $\frac{12 \cdot 11}{2} = 66$

Section 17.3

1. (a) Yes (b) No (c) Yes

2. Yes: $A \leftrightarrow 3$, $B \leftrightarrow 4$, $C \leftrightarrow 2$, $D \leftrightarrow 5$, $E \leftrightarrow 1$

3. 11 different graphs, 6 of which are connected:



4. Only if the subgraph is equal to the whole graph, because the number of vertices and the number of edges in two isomorphic graphs must be respectively the same.

8. For example: call two finite sets equivalent if they have the same number of elements.

Section 17.4

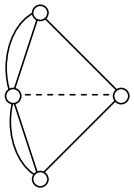
1. $\frac{d_1 + d_2 + ... + d_n}{2}$. We divide by two because each edge connects two vertices and will be counted twice.

2. No: the number of edges would be $\frac{2+2+4+3+6}{2} = 8.5$.

5. Each vertex in the graph must be connected to all $n-1$ other vertices, so the graph is isomorphic to K_n.

6. Start from any vertex, call it A_1. Follow one of the edges from A_1 to the next vertex, call it A_2. Follow from A_2 to the next vertex, A_3. And so on. Because the graph has a finite number of vertices, at some point you will return to a vertex that has been visited before: $A_{k+1} = A_i$. But, if $i > 1$, A_i is already connected to A_{i-1} and A_{i+1}, and its degree is 2, so it can't also be connected to A_k. Therefore, $i = 1$ and $A_{k+1} = A_1$ — you have returned to the starting point. The subgraph with vertices $A_1, A_2, ..., A_k$ and the edges that connect them is isomorphic to C_k. There are no other edges that come out of these vertices because the degree of each vertex in the graph is 2. If we start from any other vertex, *B*, we can trace a cycle that goes through *B*, and this cycle cannot intersect with the first. (In fact, belonging to the same cycle is an equivalence relationship for the vertices of our graph.) So, our graph must be a union of disjoint cycles. But we know that it is connected, so it must consist of only one cycle. We have to conclude that $k = n$, and our graph is isomorphic to C_n.

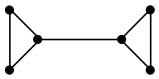
10. In the Bridges of Königsberg puzzle, all four vertices have an odd degree, and removing any edge will make the puzzle solvable. If we don't want to begin or end the path on the island, it is better to remove one of the edges that goes out of the "island" vertex:



12. $L = 0$ or $L = 2$

14. For example, C_n is such a graph.

16.

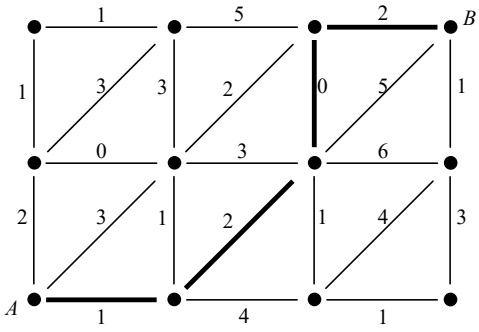


18. No. Such a circuit would have to cross each of the three "bridges" (horizontal edges) once, but it must go from left to right and back an even number of times.

Section 17.5

1. Two directed graphs are called isomorphic if there exists a one-to-one correspondence between their vertices and a one-to-one correspondence between their directed edges, such that an edge goes from vertex *A* to vertex *B* in the first graph if and only if the corresponding edge goes from the vertex that corresponds to *A* to the vertex that corresponds to *B* in the second graph.

3.



6. A directed graph has an Euler circuit if and only if for each vertex the number of arrows coming in is equal to the number of arrows going out.

Section 17.6

1.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

3.

$$C_5 \qquad\qquad K_5$$

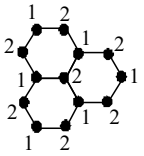
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

4. Only (d)

7. If $B = A^2$, the element b_{ij} in B is 1 if and only if the i-th and j-th vertices are connected by a path of length 2.

Section 17.7

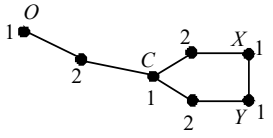
1.



3. The endpoints of a path of odd length must be different colors; the endpoints of a path of even length must be the same color.

5. The necessary and sufficient condition for a graph to be colorable in two colors is that the graph does not contain any odd-length cycles. This condition is necessary because if a graph contains an odd-length cycle, we can't properly color that cycle in two colors, let alone the whole graph.

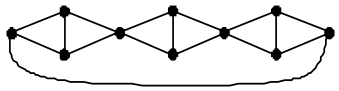
   To show that this condition is sufficient, we need to show how to properly color in two colors any graph with no odd-length cycles. Take any vertex *O* and color it black. For any vertex *X* in the graph consider the shortest path from *X* to *O*. If its length is even, color *X* black; otherwise color *X* red. Let us show that this coloring is proper. Take two vertices, *X* and *Y*, connected by an edge. Suppose they are colored in the same color. Then their distances from *O* have the same parity. Suppose *XO* and *YO* are the shortest paths from *X* to *O* and from *Y* to *O*, respectively. These paths first go separately, then they can meet at a vertex *C* (or *O* itself). For example:



   The lengths of the paths *XC* and *YC* also have the same parity. These paths, combined with the *XY* edge, would form an odd cycle. But we know our graph does not have such cycles.

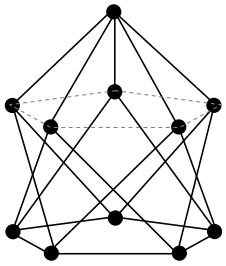
   We have shown that a graph is properly colorable in two colors if and only if it does not contain any odd-length cycles. Q.E.D.

8.

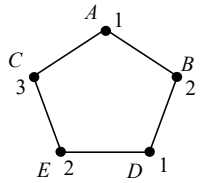


   You can actually replace any edge with a chain of "rhombuses," like the one shown in the solution to Question 7.

9. This graph can be constructed in the shape of a three-layer "wedding cake" or a "lampshade":



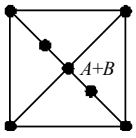
The bottom layer is simply a pentagon (C_5). Note that if a pentagon is properly colored in three colors, and you consider pairs of neighbors for each vertex, you will encounter all possible combinations of two colors among them. For example, in this colored graph —



— *A* and *E*, the neighbors of *C*, give the 1-2 pair; *C* and *D*, the neighbors of *E*, give the 1-3 pair; *C* and *B*, the neighbors of *A*, give the 2-3 pair. (It is fairly easy to show that this is true for any polygon with an odd number of sides.) This fact gives us a hint as to how to construct the middle layer. Let's make it another pentagon, above the first one. However, we won't use its sides, only its five vertices. We connect each vertex to the two neighbors of the vertex below it. We don't form any triangles when we do that. To properly color the bottom and the middle layer, we will need to use all three colors in the middle layer. For example, if a pair of neighbors is colored 1-2, the vertex above should be colored 3. The final top layer consists of only one vertex. We connect it to all five vertices of the middle layer. There is no free color left for the top vertex, so this graph cannot be properly colored in three colors.

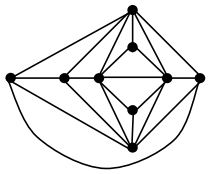
Section 17.8

2.



3. Consider a subgraph that includes the triangle and all the vertices inside. It has fewer vertices than our graph, so it can be properly colored in *p* colors. Now consider the subgraph that includes the same triangle and all the vertices outside. It, too, can be properly colored in *p* colors. We can rotate the colors in the second coloring in such a way that the three different colors used for the vertices of the triangle become the same as in the first coloring. Now the colorings match on the border (our triangle), and we can put them together to color the original graph.

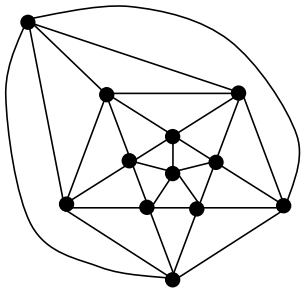
4. For example:



5. When we say "Let's take any two vertices that are not connected by an edge..." we forget that such vertices may not exist. Our graph may be K_n ($n \geq 4$). All we proved is that if the graph cannot be colored in three colors, our "gluing" process eventually reduces it to K_n.

6. We assume the graph is fully triangulated; if it isn't, we temporarily add a few edges. When they are removed, the degree of any vertex can only decrease. $E = \frac{3R}{2}$, because each region is bounded by three edges, and each edge is shared by two regions. If the degree of every vertex were greater than or equal to 6, we would have $E \geq \frac{6V}{2}$, because the number of edges meeting at each vertex would be greater than or equal to 6, and each edge connects two vertices. Then we would have $R = \frac{2E}{3}$ and $\frac{E}{3} \geq V$. From Euler's formula, $V - E + R = 2 \;\Rightarrow\; \frac{E}{3} - E + \frac{2E}{3} \geq 2 \;\Rightarrow\; 0 \geq 2$ — a contradiction. Our assumption that the degree of every vertex is greater than or equal to 6 cannot be true. Q.E.D.

7.



The smallest number of vertices in such a graph is 12. If k is the number of vertices of degree 5 and the rest have a degree greater than or equal to 6, we would get $E \geq \frac{6(V-k)+5k}{2} = \frac{6V}{2} - \frac{k}{2} \;\Rightarrow$ $V \leq \frac{E}{3} + \frac{k}{6}$ (see the previous question). Euler's formula would give us $\frac{E}{3} + \frac{k}{6} - E + \frac{2E}{3} \geq 2 \;\Rightarrow\; \frac{k}{6} \geq 2 \;\Rightarrow\; k \geq 12$.

18 Number Theory and Cryptology

Section 18.2

1. If $d \mid a$ and $d \mid b$, then $d \mid a - b$. On the other hand, if $d \mid a - b$ and $d \mid b$, then $d \mid \big((a - b) + b\big)$, that is, $d \mid a$. So, a, b and $a - b, b$ have the same common divisors.

3.
```
def gcd(a, b):
    """Return the greatest common divisor of a and b."""
    if a > b:
        return gcd(a - b, b)
    elif a < b:
        return gcd(a, b - a)
    else:
        return a     # or return b
```

This code is also included in `Py\PythonCode\`**`NumberTheory.py`**.

4.
```
def gcd(a, b):
    """Return the greatest common divisor of a and b."""
    while a > 0:
        b, a = a, b % a
    return b
```

This code is also included in `Py\PythonCode\`**`NumberTheory.py`**.

5. If *a* and *b* are relatively prime, the equation $ax + by = 1$ has a solution (x_0, y_0). Then (cx_0, cy_0) is a solution of $ax + by = c$.

10. Let $c, c + p, c + 2p, \ldots$ be an arithmetic sequence. Since *p* and *q* are two different primes, they are relatively prime, and the equation $qx - py = c$ has a solution (x_0, y_0). $(x_0 - kp, y_0 + kq)$ is also a solution for any *k*, so there is a solution (x, y) with $y > 0$. Then $qx = c + py$ is a term of our arithmetic sequence that is divisible by *q*.

13. The equation $ax + by + cz = 1$ has a solution if and only if GCD(a, b, c) = 1.

1. Let GCD(a, b, c) = d. $d \mid a$, $d \mid b$, and $d \mid c$, so $d \mid ax + by + cz$. Therefore, if $d \neq 1$, the equation $ax + by + cz = 1$ cannot have solutions.

2. Now let's suppose that $d = 1$ and show that the equation $ax + by + cz = 1$ has a solution. Let GCD(a, b) = h. h and c must be relatively prime (if they had a common divisor, it would be a common divisor of a, b, and c). So, the equation $hu + cz = 1$ has a solution (u, z). Since GCD(a, b) = h, $a = ha_1$ and $b = hb_1$, where a_1 and b_1 are relatively prime. So, the equation $a_1x + b_1y = u$ has a solution (x, y). We get
$hu + cz = 1 \Rightarrow h(a_1x + b_1y) + cz = 1 \Rightarrow ha_1x + hb_1y + cz = 1 \Rightarrow ax + by + cz = 1$.

Section 18.3

4. `1*2*...*13 + 1 = 30031 is not a prime.`

The above is the output from the following program:

```
n = 1
p = 2
while p <= 1000:
    if is_prime(p):  # is_prime is described in Section 7.4
        n *= p
        if not is_prime(n+1):
            print('1*2*...*{0:d} + 1 = {1:d} is not a prime.'.format(p, n+1))
            break
    p += 1
if p > 1000:
    print('Looks like all are primes.')
```

If you want to know what the prime factors of 30031 are, use the program from Question 3.

This code is also included in `Py\PythonCode\`**`NumberTheory.py`**.

6. This is in fact true for any odd integer. If p is odd, both p+1 and p−1 are even.
$$p = \left(\frac{p+1}{2}\right)^2 - \left(\frac{p-1}{2}\right)^2.$$

7. Fibonacci numbers get pretty big fast: for example, $F_{100} = 354,224,848,179,261,915,075$. A naive approach in which we generate Fibonacci numbers in sequence and check each for being a prime is not practical — it will run forever. A better approach would be to use a standard iterative function for Fibonacci numbers and test F_{100}, F_{99}, and so on, until we find a prime. We can try it for finding the largest Fibonacci prime among the first n Fibonacci numbers with n equal to, say, 25. But this method will still run forever for n = 100, because testing a large number for primeness is not a trivial task. Luckily, someone has already created a list of Fibonacci primes — see, for example, http://en.wikipedia.org/wiki/List_of_prime_numbers#Fibonacci_primes or https://oeis.org/A005478 or google "fibonacci primes" As we can see, the largest Fibonacci prime that does not exceed F_{100} is 99,194,853,094,755,497.

10. Each divisor d can be expressed as $d = p_1^{c_1} \cdot ... \cdot p_k^{c_k}$, where $0 \le c_1 \le j_1,\ 0 \le c_2 \le j_2,\ ..., 0 \le c_k \le j_k$. The number of possible values for c_1 is $j_1 + 1$, for c_2 is $j_2 + 1$, and so on. The total number of divisors is $(j_1 + 1) \cdot (j_2 + 1) \cdot ... \cdot (j_k + 1)$.

11. Integers that are not relatively prime with n are divisible by p or by q (or by both). The number of positive integers below n that are divisible by p is $\frac{n}{p}$. The number of positive integers below n that are divisible by q is $\frac{n}{q}$. The number of positive integers below n that are <u>not</u> relatively prime with n is $\frac{n}{p} + \frac{n}{q} - \frac{n}{pq}$ (we need to subtract $\frac{n}{pq}$ because we have counted the numbers divisible by both p and q twice). The number of positive integers below n that are relatively prime with n is $n - \left(\frac{n}{p} + \frac{n}{q} - \frac{n}{pq}\right) = n\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right)$.

    If $n = p_1^{j_1} \cdot ... \cdot p_k^{j_k}$, then the number of positive integers below n that are relatively prime with n is $n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdot ... \cdot \left(1 - \frac{1}{p_k}\right)$. This expression is called *Euler's totient function* and is commonly denoted as $\varphi(n)$. The proof is similar.

Section 18.4

3.

```
def elapsed_time(hour1, min1, hour2, min2):
    return hour2 * 60 + min2 - hour1 * 60 - min1
```

4.

```
def thanksgiving(jan1):
    november_1 = (january_1 + 304) % 7
    first_thursday = 5 - november_1
    if first_thursday <= 0:
        first_thursday += 7
    return first_thursday + 21
```

Or, simply,

```
def thanksgiving(january_1):
    return (2 - january_1) % 7 + 21
```

5.

+	**0**	**1**	**2**	**3**	**4**	**5**
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

*	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

8. $n \equiv -1 \pmod 3$, $n \equiv -1 \pmod 4$, $n \equiv -1 \pmod 5$, and so on, $n \equiv -1 \pmod{12}$. Therefore, the answer is the least common multiple of 3, 4, 5, ..., 12 minus 1, which is 27719.

9. $3^{22222} = \left(3^{1010}\right)^{22} \cdot 3^2$. By Fermat's Little Theorem, $\left(3^{1010}\right)^{22} \equiv 1 \pmod{23}$. Therefore, $3^{22222} \equiv 3^2 \pmod{23} \equiv 9 \pmod{23}$.

11. $p^2 - 1 = (p-1)(p+1)$. p is a prime and it is not 2 or 3, so it is odd and not divisible by 3. $p-1$ and $p+1$ are two consecutive even numbers, so one of them must be divisible by 4. Therefore, their product must be divisible by 8. Also, $p-1$, p, and $p+1$ are three consecutive integers, so one of them must be divisible by 3. It can't be p, so it is either $p-1$ or $p+1$. $(p-1)(p+1)$ is divisible by 8 and by 3, so it must be divisible by 24.

14. (b) Any number m, such that $0 < m < p$, has a reciprocal modulo p. Only 1 and $(p-1)$ are their own reciprocals. So all the factors in $(p-1)!$, except 1 and $(p-1)$, can be split into pairs such that the product of the numbers in each pair is 1 (mod p). Therefore, $(p-1)! \equiv p-1 \pmod p$ and $(p-1)! + 1 \equiv 0 \pmod p$, that is, $(p-1)! + 1$ is divisible by p.

Section 18.5

2.

```
abc = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
def encode(text, key):
    """Encode text using Vigenere cyper.
        text contains only uppercase letters."""
    len_key = len(key)
    code = ''
    k = 0

    for c in text:
        i = abc.find(c)
        j = abc.find(key[k])
        c = abc[(i + j) % 26]
        code += c
        k = (k + 1) % len_key

    return code

def decode(code, key):
    """Decode code encoded with Vigenere cyper."""
    len_key = len(key)
    text = ''
    k = 0

    for c in code:
        i = abc.find(c)
        j = abc.find(key[k])
        c = abc[(i - j) % 26]
        text += c
        k = (k + 1) % len_key

    return text
```

```
key = 'LEMON'
text = 'ATTACKATDAWN'

print('<' + text + '>')
code = encode(text, key)
print('<' + code + '>')
text = decode(code, key)
print('<' + text + '>')
```

Output:

```
<ATTACKATDAWN>
<LXFOPVEFRNHR>
<ATTACKATDAWN>
```

This code is also included in `Py\PythonCode\`**`Cipher.py`**.

4.

```
p = 170141183460469231731687303715884105727
a = 61897001964269013744956211l
r = 5
print(pow_mod(r, a, p))
```

Output:

```
263284389788069415466160713518247260771
```

This code is also included in `Py\PythonCode\`**`NumberTheory.py`**.

6. We set $D = x$, where (x, y) is a solution of the Diophantine equation $Ex - (p-1)(q-1)y = 1$. $p = 13$, $q = 17$, and $E = 5$. $5x - 12 \cdot 16y = 1 \Rightarrow 5x - 192y = 1 \Rightarrow x = 77,\ y = 2$. $D = 77$.

7. Alice locks a message in a box with her lock and sends the box to Bob. Bob adds his lock to the box and sends the box back to Alice. Alice receives the box, now locked with two locks, removes her lock, and sends the box back to Bob. Bob removes his lock and reads Alice's message.