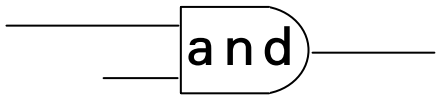
Mathematics
for the Digital Age



Programming
in Python

>>> Second Edition:
with Python 3

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: 2009913596

ISBN 978-0-9824775-8-8

1 2 3 4 5 6 7 8 9 10 15 14 13 12 11 10

Printed in the United States of America

10 Parity, Invariants, and Finite Strategy Games

10.1 Prologue

Suppose you are buying a box of cereal in a grocery store. The cashier runs the box through the scanner, which reads the UPC (Universal Product Code) barcode from it. Sometimes an error occurs, and the cashier has to scan the same item again. But how does the system know that an error has occurred? If it read the UPC incorrectly, it could potentially charge you for a can of tuna instead of the box of cereal. Well, it turns out that not all 12-digit numbers are valid UPCs. In fact, if you change any one digit in a UPC, you will get an invalid code that does not match any product. This is because not all digits in a UPC carry information: the last digit is the *check digit*, which depends on the previous digits. If you change one digit in a UPC, the check digit no longer matches the code. We can say that a UPC has built-in *redundancy*: the information is not represented with optimal efficiency. The check digit is redundant because it can be computed from the other digits. Redundancy allows us to detect and sometimes even correct errors.

When we store or transmit binary data, we can stipulate that the number of 1's in each byte be even (or odd). Only seven bits in each byte would then carry information; the eighth bit would be used as a kind of check digit. It is called the *parity bit*. We will discuss parity and check digits in Section 10.2.

Suppose the data is encoded in bytes with *even parity*. This means the total number of bits set to '1' in each byte is even. If a quantity or property remains constant throughout a process, such a quantity or property is called an *invariant*. The sum of the angles in any triangle is 180 degrees. This is an invariant. If within a `while` loop you add 1 to `m` and subtract 1 from `k`, after each iteration the sum `m + k` remains unchanged. This is an invariant. We will talk about invariants and their role in mathematics and computer science in Section 10.3.

A *finite strategy game* for two players is a game with a finite number of possible positions. Two players take turns advancing from one position to the next, according to the rules of the game. Positions never repeat; that is, it is not possible to return to a position visited earlier. Some of the positions are designated as winning positions: the player who reaches a winning position first wins. If neither of the players can

make a valid move and neither has reached a winning position, then it is a tie. Tic-tac-toe is an example of a finite strategy game.

In strategy games of this kind, either the first or the second player always has a winning strategy, or both have a strategy that leads to a tie. In games where a tie is not possible, it is often desirable to describe the winning strategy using some invariant — a property shared by all the winning positions, and by all *safe* positions where the winning player can land. The losing player is always forced to abandon a safe position and move into an *unsafe* one. Nim is an example of such a game. We will discuss the details and see some examples of finite games and their strategies, including Nim, in Section 10.4.

10.2 Parity and Checksums

If you store binary data or send information over a communication line, some errors might occur. A simple method for detecting errors is based on *parity*. Usually the data is divided into relatively short sequences of bits, called *codewords* or *packets*.

> **The term *parity* refers to the evenness or oddness of the number of bits set to '1' in a packet. If this number is even, we say that the packet has *even parity*; otherwise we say it has *odd parity*.**

When a chunk of data is transmitted, the transmitter computes the parity of data and adds one bit (called *parity bit*), so that the total parity of the packet including the parity bit is even. Then, if the receiver gets a packet with odd parity, it reports an error. (The transmitter and receiver may "agree" to use odd parity instead of even parity.) When parity is used, all data packets usually have the same length. For example, seven data bits plus one parity bit or 31 data bits plus one parity bit.

Example 1

The following sequence of seven-bit codes encodes the word "parity" in ASCII:

```
1110000 1100001 1110010 1101001 1110100 1111001
```

We want to add a parity bit to each code so that it gets even parity. What are the resulting eight-bit packets?

Solution

```
11100001 11000011 11100100 11010010 11101000 11110011
```

❖ ❖ ❖

It can happen, of course, that two errors occur in the same packet — two bits are flipped from 0 to 1 or from 1 to 0. Then the parity of the packet remains unchanged, and the error goes undetected. The parity method relies on the assumption that the likelihood of two errors in the same packet is really low. If errors are frequent, then a small number of bits require a parity bit for error checking. The more reliable a communication channel or a storage system is, the longer the data packets that can be used.

❖ ❖ ❖

If we swap two consecutive bits in a data packet, its parity does not change. Luckily such *transposition errors* are very rare when the packet is generated by a computer or another device. Not so with us humans. When we type words or numbers, transposition errors are common. So parity-type error detection does not work well when humans are involved. For example, when a cashier gives up on the scanner that cannot read the UPC from a crumpled bag of chips, he enters it manually. The cashier may mistype a digit or transpose two digits. There must be a mechanism that detects such errors. Such a mechanism uses *checksums* and *check digits*.

Example 2

Driver's licenses on the island of Azkaban have six digits. The sixth digit is the check digit: it is calculated as follows: we add up the first five digits, take the resulting sum modulo 10 (the remainder when the sum is divided by 10), and subtract that number from 10. For example, if the first five digits of a driver's license are `95873`, the check digit is 10 - ((9 + 5 + 8 + 7 + 3) mod 10) = 10 - (32 mod 10) = 10 - 2 = 8. So `958738` is a valid driver's license number on Azkaban. Note that if we add all six digits and take the result modulo 10, we get 0. Does this system detect all single-digit substitution errors? All transposition errors?

Solution

This system detects all single-digit substitution errors, because if you change one digit, the sum of the digits modulo 10 is no longer 0. However, the sum does not depend on the order of digits, so transposition errors are not detected.

❖ ❖ ❖

To detect both substitution and transposition errors we need a slightly more elaborate algorithm for calculating the check digit. We can multiply some of the digits by certain coefficients before adding them to the sum. Several real-world checksum algorithms are described in the exercises.

Exercises

1. Which of the following bit packets have even parity? Odd parity?
(a) `01100010` (b) `11010111` (c) `10110001`. ✓

2. How many bytes (all possible eight-bit combinations of 0s and 1s) have even parity? Odd parity? ✓

3. Section 8.5 describes the n-choose-k numbers $\binom{n}{k}$. For any given $n \geq 1$, the sum of $\binom{n}{k}$ for all even k is the same as the sum of $\binom{n}{k}$ for all odd k. For example, for $n = 4$, $\binom{4}{0}+\binom{4}{2}+\binom{4}{4}=\binom{4}{1}+\binom{4}{3}$. Indeed, $1+6+1=4+4$. Why is this true for any $n \geq 1$? ⩽ Hint: see Question 2. ⩾ ✓

4. Write and test a Python function that takes a string of binary digits and returns its parity (as an integer, 0 for even, 1 for odd).

5. Consider the following table of binary digits:

```
0 1 1 0 1 1
1 0 0 0 1 0
1 0 1 0 0 1
0 1 0 1 0 0
```

It was supposed to have even parity in all rows and all columns, but an error occurred and one bit got flipped. Which one? ✓

6.■ Write and test a Python function `correctError(t)`, which takes a table, such as described in Question 5 (but not necessarily 4 by 6), with a possible single-bit error, checks whether it has an error, and, if so, corrects it. The table `t` is represented as a list of its rows; each row is represented as a string of 0s and 1s (all the strings have the same length). For example, the table from Question 5 would be represented as

```
['011011', '100010', '101001', '010100']
```

7.■ How many 4 by 6 tables of binary digits have even parity in all rows and all columns? Odd parity? ✓

8.■ The UPC has 12 decimal digits. The checksum is calculated as follows: we take the sum of all the digits in odd positions, starting from the left (first, third, fifth, etc.), multiply it by 3, then add the sum of all the digits in even positions (second, fourth, etc.). In a valid UPC, the checksum must be evenly divisible by 10. For example, `072043000187` is a valid UPC, because $0 \cdot 3 + 7 + 2 \cdot 3 + 0 + 4 \cdot 3 + 3 + 0 \cdot 3 + 0 + 0 \cdot 3 + 1 + 8 \cdot 3 + 7 = 60$, which is evenly divisible by 10. Write and test a Python function that takes a string of 12 digits and returns `True` if it represents a valid UPC; otherwise it returns `False`.

9. Does the checksum method for UPCs, described in Question 8, detect all single-digit substitution errors?

10.■ In the UPC checksum algorithm, described in Question 8, the odd-placed digits are multiplied by 3. Why 3 and not 2? ✓

11.◆ Does the checksum method for UPCs, described in Question 8, detect all transposition errors?

12.■ Credit card numbers have 16 digits. The checksum is calculated as follows. Each digit in an odd position (first, third, etc.) is multiplied by 2. If the result is a two-digit number, its two digits are added together; otherwise it is left alone. The result is added to the sum. Each digit in an even position (second, fourth, etc.) is added to the sum as is. The resulting sum must be 0 modulo 10. For example, `4111111111111111` is a valid credit card number: its checksum is 30. `4111111111111178` is another valid number: its checksum is 40. Write and test a Python function that checks whether a given string of 16 digits represents a valid credit card number. Come up with a few other valid numbers and use a few invalid numbers for testing.

13.■ The book industry uses ISBNs (International Standard Book Numbers) to identify books. In 2007 the industry switched from 10-digit ISBNs to 13-digit ISBNs. The last digit in ISBN-10 and in ISBN-13 is the check digit. But the algorithms for calculating the check digit are different for ISBN-10 and ISBN-13.

In ISBN-13, the check digit is calculated the same way as in UPC, only the positions of the digits are counted starting from 0. The first digit is added to the sum as is, the second digit is multiplied by 3, the third digit is added as is, the fourth is multiplied by 3, and so on.

ISBN-13 is obtained from ISBN-10 by appending '978' at the beginning and recalculating the check digit.

Write a Python function `isbn13CheckDigit` that calculates and returns the ISBN-13 check digit (a single character) for a given string of 12 digits, and another function, `isbn10to13`, that converts ISBN-10 (a string of 10 digits) to ISBN-13 and returns a string of 13 digits. Test your functions thoroughly. Use these test data, for example:

ISBN-10	ISBN-13
0982477503	9780982477502
0982477511	9780982477519
098247752X	9780982477526
0982477538	9780982477533
0982477546	9780982477540
0982477554	9780982477557
0982477562	9780982477564
0982477570	9780982477571
0982477589	9780982477588
0982477597	9780982477595

14.■ Question 13 asks you to write two functions that help convert ISBN-10 to ISBN-13. Now write two similar functions to convert ISBN-13 to ISBN-10. In ISBN-10, the check digit is calculated as follows. The first digit is multiplied by 1, the second by 2, the third by 3, and so on; the ninth digit is multiplied by 9. The products are added together and the result is divided by 11 with the remainder. The remainder is used as the check digit. If the remainder is 10, the letter 'X' is used as the check "digit."

10.3 Invariants

If I have several lollipops and give you some, and you give some to Candy, and she gives some back to me and you, the total number of lollipops among the three of us remains the same (as long as we don't eat any) — it is an *invariant*. If a particle moves along a circle, its distance from the center of the circle remains constant — it is an invariant. If a bishop moves on the chessboard along a north-east to south-west diagonal, the sum of the bishop's row and column positions, counting from the upper-left corner, remains the same — it is an invariant. The concept of invariant is useful in physics, in mathematics, and in computer science.

Example 1

If you toss a small rock in the air, its energy consists of the kinetic energy $\frac{mv^2}{2}$ and the potential energy mgh, where m is the mass of the rock, v is its speed, h is the height above ground, and g = 9.8 m/sec^2 is the acceleration due to gravity. How far up above the ground will the rock fly if it was tossed straight up from ground level with an initial velocity of 20 m/sec?

Solution

$\frac{mv^2}{2} + mgh$ is an invariant, it remains constant. So $\frac{mv_0^{\,2}}{2} + mgh_0 = \frac{mv_1^{\,2}}{2} + mgh_1$

At the beginning, $h_0 = 0$, $v_0 = 20$ m/sec. At the top, $v_1 = 0$. Comparing the total energy at the ground and at the top, we get $\frac{mv_0^{\,2}}{2} = mgh_1 \;\Rightarrow\; h_1 = \frac{v_0^{\,2}}{2g} \approx 20.4$ meters.

In mathematics, invariants are ubiquitous. One of the applications is in strategy games.

Example 2

There is a round table in a room and three bags of coins: one with quarters, one with dimes, and one with nickels. Two players take turns, picking one coin from any bag and placing it anywhere on the table, without overlapping any other coins. There are enough coins in each bag to cover the entire table. The player who places the last coin, leaving no space for more coins, wins. Does the first or a second player have a winning strategy?

Solution

In this game, there is an infinite number of possible configurations of coins on the table. But the game always ends after several moves, because only a finite number of coins fit on the table. There are no ties.

One approach to finding a strategy in games of this type is to come up with a clever invariant condition, a kind of "balance," which the winner can maintain, always moving into a safe position where the condition is satisfied and forcing the opponent to abandon this safe position, to lose "balance." In this particular game, symmetry with respect to the center of the table comes to mind as a useful invariant. The first player can establish symmetry by placing the first coin at the center of the table. On subsequent moves, the first player always picks the same size coin as the opponent and places it symmetrically to the one just placed by the second player. The first player always maintains the symmetry of the configuration with respect to the center of the table, an invariant. The second player is always forced to break the symmetry. At the end, the first player places a coin in the last remaining spot.

❖ ❖ ❖

In computer science, the concept of invariant applies to loops. A condition that is related to the purpose of the computation and holds true before the loop and after each iteration through the loop is called a *loop invariant*. Loop invariants are useful for reasoning about the code and for formally proving its correctness.

Example 3

The code below (from Figure 2-2 on Page 25) has one loop:

```
def sum1ToN(n):
    "returns 1 + 2 + ... + n"
    s = 0
    k = 1
    while k <= n:
        s += k # add k to s
        k += 1 # increment k by 1
    return s
```

Find a loop invariant for that loop.

Solution

The purpose of the loop is to calculate $1+2+...+n$. Before the loop, $s=0$ and $k=1$. After the last iteration through the loop, $k=n+1$ and $s=1+2+...+n$. The loop invariant here is $s=1+2+...+(k-1)$.

Exercises

1. A domino covers exactly two squares on a chessboard, so it is possible to cover the board with 32 dominos. Now suppose we cut out the two white squares at the opposite corners of the board. Try to cover the remaining 62 squares with 31 dominos. Is it possible? If not, explain why not. ✓

2. In chess a knight moves by two squares in one direction, then by one square in a perpendicular direction. Is it possible for a knight to visit each square exactly once and return to the starting position? If yes, show an example; if not, explain why not. Is it possible on a 7-by-7 "chessboard"?

3.■ Consider a rectangle *AOBC* on the coordinate plane, such that *O* is the origin, *AO* is on the *x*-axis, *OB* is on the *y*-axis, and *C* is in the first quadrant. Describe the locus of points (that is the set of all points) *C*, such that the perimeter of the rectangle is equal to *p*, a constant.

4.■ Consider a rectangle *AOBC* in the first quadrant on the coordinate plane, such that *O* is the origin, and *C* moves along a branch of the hyperbola $y = \frac{1}{x}$. Describe an invariant property of the rectangle (beyond the obvious fact that *O* stays at the origin). ✓

5.■ Demonstrate geometrically that among all the rectangles with a given area, the square has the smallest perimeter. ≤ Hint: see Questions 3 and 4. ≥ ✓

6.■ Several pluses and minuses are written in a line, for example: $+---++--++++-$. If the first two symbols at the beginning of the line are the same, you add a plus at the end; if they are different, you add a minus. Then you erase the first two symbols. The operation is repeated until only one plus or minus remains. Is the remaining symbol always the same, regardless of whether you have proceeded from left to right or from right to left?

7.■ Erin and Owen share a computer. They want to make a schedule for the exclusive use of the computer, from noon to midnight, and they have decided to turn this into a game. On each move, a player can reserve a contiguous block of available time, up to two hours, starting at any time. The players take turns making reservations. They have tossed a coin, and Erin goes first. Does Owen have a strategy that would allow him to get at least as much total time as Erin, no matter what she does?

8.◆ The table below lists the numbers of vertices, edges, and faces in four polyhedrons:

	Vertices	**Edges**	**Faces**
Tetrahedron	4	6	4
Cube	8	12	6
Triangular prism	6	9	5
Icosahedron	12	30	20

Describe an invariant that connects the number of vertices *V*, the number of edges *E*, and the number of faces *F* in a polyhedron. Show that this is indeed an invariant for all polyhedrons. In fact, the edges and faces do not have to be "straight:" the same invariant remains as long as the edges and faces do not intersect. ✓

9. Identify a loop invariant for your solution to Question 6 in Section 4.5 (page 75).

10.4 Finite Strategy Games

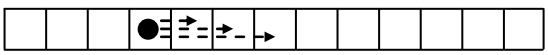
Suppose a game has a finite number of possible positions. Two players take turns advancing from one position to the next, according to the rules of the game. Positions never repeat: a position that occurred once can never happen again. So sooner or later, when a *terminal* position is reached, the game ends. Depending on the rules, the player who made the last move wins or loses, while some of the terminal positions may be designated as ties. Alternatively, the players may collect some points along the way, and the player with the higher score wins. Games of this type are called *finite strategy games*. In some games, such as Nim, reaching a terminal position (taking the last stone) signifies a win, and there are no ties. We will discuss real Nim a little later; first, let us consider a very simple version.

Example 1

There is a pile of N stones. Two players take turns making moves. On each move, a player can take one, two, or three stones from the pile. The player who takes the last stone wins. Let's call this game Nim-1-3 (*nimm* is German for "take"). Does the first or the second player have a winning strategy? What is that strategy?

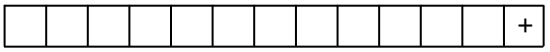
Solution

This game is equivalent to (a mathematician might say *isomorphic*, that is, "has the same form" as) the game where the players advance a token along a linear board from left to right; on each move a player can advance the token by one, two, or three squares:

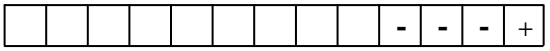


The length of the board is $N+1$. The position on the board corresponds to the number of stones left in the pile; the first square corresponds to N stones, the last one to 0 stones in the pile. (It is often convenient to choose a model for the game, which is isomorphic to the original game, but is easier to visualize and work with.)

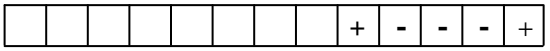
To solve this game, we will use a work-backwards method. Let us first mark the winning position with a plus sign:



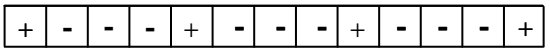
This position is safe: that's where you want to end up. Now let us find all the positions from which the plus position can be reached. These are unsafe for you: if you land there, your opponent can jump to a plus. Let us mark each of them with a minus sign:



Now there is one position from which you can move only to a minus (unsafe position). This position is safe for you, so let's mark it with a plus:



Again let us find all the positions from where one can reach any plus position and mark them with minuses. Continuing this process, we will eventually mark up the whole board:



If the starting position is safe (marked with a plus), the first player is forced to abandon it, and the second player can win, moving to a safe position on every turn. In Nim-1-3 this happens when the initial number of stones $N = 4k$ (N is evenly divisible by four). If the starting position is unsafe (marked with a minus), then the first player can move to a safe position right away, and eventually win. In Nim-1-3 this happens when N is not evenly divisible by four.

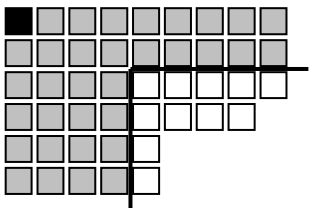
❖ ❖ ❖

We can represent all the positions in a finite game as points and each legal move as an arrow from one position to the next. As explained later, in Chapter 15, such a structure is called a *directed graph*. The fact that game positions never repeat means that the graph is *acyclic*; that is, it has no circular paths. Representing games as graphs is useful for developing the mathematical theory of finite games. A lot is known about graphs! But it is too cumbersome for analyzing a particular game, even as simple as Nim-1-3. It is better, instead to look for a simple enough property or formula that describes all the safe positions.

There is a simple formula for the safe positions in the Nim-1-3 game: it is safe to move to a position where the number of stones remaining in the pile is evenly divisible by 4. So there is no need to remember the chart of all safe positions; it is much easier to use the formula. Not every game, however, has a simple formula. Let us consider a more interesting game, for which a simple property or formula for safe positions hasn't been found yet.

Example 2

In the game of Chomp, the board represents a rectangular grid (like a bar of chocolate). Players take turns taking "bites" from the grid. A "bite" consists of one square plus all the squares to the right and/or below it. For example:



The square in the upper left corner is "poison"; the player who is forced to "eat" the poison loses the game.

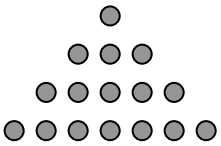
It is possible, of course, to apply to Chomp the work-backwards method described in Example 1 and come up with a list of all safe positions (see Question 8 in the exercises). Such analysis, however, would be very tedious for larger boards if we tried doing it by hand. It is better to program a computer to do that. No one, so far, has been able to come up with a compact property or formula that describes all safe positions in Chomp, except for two special board sizes: *n*-by-2 and *n*-by-*n* (see Questions 6 and 7 in the exercises).

One peculiar thing about Chomp is that we can prove, even without knowing anything about any specific strategy, that the first player can always win (as long as the board is larger than 1 by 1). Here is a proof by contradiction. Suppose it is the second player who has a winning strategy. So he has a winning response to each of the first player's moves. If the first player "bites off" just one square, at the lower right corner, in his first move, the second player has a winning move in response. But the first player could have made that winning move first! This proof is based on the argument called *strategy stealing*: in this game the second player can't have a winning strategy because the first player would "steal" it.

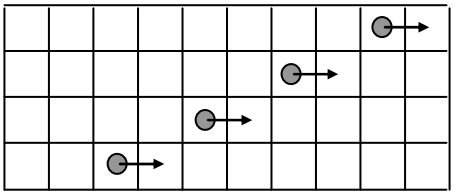
❖ ❖ ❖

Our last example in this section is the real game of Nim. In Nim there are several piles of stones. On each move, a player must take at least one stone but can take any number of stones from one pile. The player who takes the last stone wins. (There is another version of Nim in which the player who takes the last stone loses. The winning strategy in this take-last-and-lose Nim is similar to our Nim.)

Nim is sometimes presented as an arrangement of cards or tokens in several rows. For example:



Each row represents a pile of stones. Another isomorphic model for Nim is several tokens on a rectangular board moving in the same direction. For example:



Eacn token's distance from the rightmost square represents the number of stones in the corresponding pile.

It is possible to use the work-backwards method to determine a strategy for Nim. This is not very interesting, though, and quite tedious when the numbers are large. Luckily, Nim has a very elegant description of the winning strategy based on an interesting property of its safe positions.

Suppose N_1, N_2, ..., N_k are the numbers of stones left in the piles. The idea is for the winning player to always maintain some kind of "balance" among these numbers. The losing player is forced to change one of the numbers and break the balance; then the winning player restores the balance again. But what kind of balance? Perhaps some kind of checksum might work. When one of the numbers changes, the checksum will change, too. The problem is, the winning player must be able to restore the checksum by reducing one of the numbers. Conventional checksum algorithms don't work like that. We need something more ingenious.

Let us arrange the binary representations of N_1, N_2, ..., N_k in one vertical column, with the rightmost digits aligned. For example, if the numbers are 1, 3, 5, and 7, we get

N_1	1
N_2	11
N_3	101
N_4	111

Now consider the parity of each column — whether the number of 1s in the column is even or odd. In the final winning position, all the numbers are zeroes, so the parity of all the columns is even. Let us declare safe all positions with this property: the parity of all columns is even. All other positions are unsafe.

The parities of the columns can be considered as binary digits of a kind of "checksum." Calculating this "checksum" is equivalent to performing bit-wise addition, or — same thing — the bit-wise XOR (exclusive OR) operation on the numbers (see Section 7.3). This "checksum" is called the *Nim sum*. In safe positions the Nim sum must be 0. For example:

	Safe:		*Unsafe:*
N_1	1	N_1	1
N_2	11	N_2	1
N_3	101	N_3	101
N_4	111	N_4	111
	===		===
Nim sum	000		010

It is very easy to calculate the Nim sum on a computer (see Question 10).

From a safe position you are forced to move to an unsafe one. Indeed, when a player takes one or several stones from the *j*-th pile, N_j changes, so at least one of its binary digits changes. The parity of the column that holds that digit will change, too. For example, in the 1-3-5-7 configuration, the numbers of bits in the three columns are 2, 2, and 4 — all even, so this is a safe position. The first player is forced to abandon it on the first move, so the second player can win.

Is it true, though, that from any unsafe position you can always move to a safe one? In other words, is it true that the even-parity-of-all-columns property can always be restored by reducing one of the numbers? The answer is yes, and here is why. Suppose some of the columns have odd parity. Let's take the leftmost of them. Since its parity is odd, it must have at least one bit in it set to 1. Let's take the row that contains that bit and flip (from 1 to 0 or from 0 to 1) all the bits in that row that are in odd-parity columns. The even parity of all columns will be restored. The new

number represented by the row will be smaller than the original number, because the leftmost flipped bit (that is, the leftmost binary digit flipped) has changed from 1 to 0.

For example, suppose in the 1-3-5-7 configuration, the first player removes the entire second pile. The numbers become 1, 0, 5, 7:

N_1	1
N_2	0
N_3	101
N_4	111
	===
Nim sum	011

The parity of the second and third columns becomes odd. The second column is the leftmost among them. The bit in the fourth row is set to 1. Let's take that row and flip the bits in it that are in the odd-parity columns. (This is equivalent to XOR-ing that row with the Nim sum.) We get:

N_1	1
N_2	0
N_3	101
N_4	**1**0**0**
	===
Nim sum	000

The even parity of all columns is restored: the Nim sum is 0 again. N_4 becomes 4. So to restore the balance and return to a safe position from the 1-0-5-7 position, the second player should take from the fourth pile 3 stones out of 7, leaving 4 stones.

❖ ❖ ❖

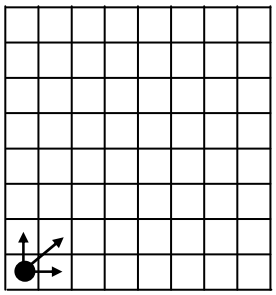
Nim is important, because there are many more general versions of it (see, for example, Question 12 in the exercises), and many games are isomorphic to a version of Nim.

Exercises

1. What property describes the safe positions in the Nim-1-4 game, in which you have one pile of stones and are allowed to take 1, 2, 3, or 4, stones on each move? ✓

2. Write a Python program that plays Nim-1-3.

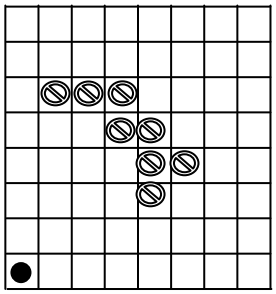
3. In this game, two players take turns moving a token on an 8-by-8 board. At the beginning, the token is placed in the lower left corner. On each move, the token can be moved by one square up or to the right or diagonally up and to the right:



The player who reaches the upper right corner first wins. Find all safe positions in this game. Can the first player always win? ✓

4. Describe an isomorphic version of the game from Question 3, as a game that uses piles of stones instead of the board. Describe the safe positions in your version. ✓

5.■ Suppose the game from Question 3 has been modified: now the field has a poisonous swamp, like this:



The player who has no valid move or is forced into the swamp loses. Find all safe positions on the above board and show that the first player can win. If the first player moves diagonally on the first move, what is the correct response?

6. Come up with the winning strategy for Chomp with an *n*-by-2 board.

7.■ Come up with the winning strategy for Chomp with an *n*-by-*n* board.

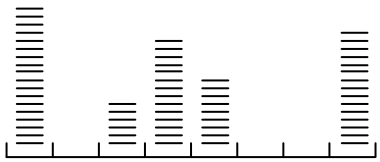
8.◆ Use the work-backwards method to find all safe positions in 4-by-3 Chomp. What is the winning first move in this game? ✓

9. Is the Nim position with four piles of 3, 4, 5, and 6 stones safe?

10. Write and test a Python function that takes a Nim position, represented as a list of non-negative integers, and checks whether it is safe.

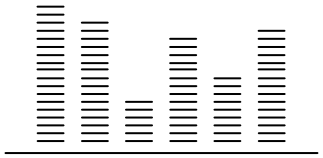
11. What is the correct move in Nim if three piles are left with 6, 8, and 11 stones in them? ✓

12.◆ Consider the following modified version of Nim. In this game, stacks of coins are placed on some of the squares of a one-dimensional board:



On his move, a player can take several coins (at least one) from any stack and add them to the next stack to the right (or start a new stack there, if that square was empty). Players are not allowed to take coins from the rightmost stack. Whoever moves the last coin wins. Describe the safe positions in this game.

13.◆ Six stacks of coins are arranged in a line on the table:



Two players take turns taking coins: on his move a player must take the whole stack of coins, either on the left or on the right end. The player who ends up with most coins wins. Come up with a strategy for the first player that assures that he collects at least as many coins as his opponent. ⦙ Hint: imagine that the stacks are arranged on a chessboard, on squares of alternating colors. ⦙ ✓

14.■ In this game there are nine cards with the numbers 1 through 9 written on them. Two players take turns, taking one card on each turn. The player who is first to collect three cards with numbers that add up to 15 wins. If the first player takes 5, is 3 or 4 the correct response? Come up with a strategy that assures a win or a tie for the first player. ⦙ Hint: imagine that the cards are arranged on the table and form a 3-by-3 magic square (the sums of the numbers in each row, each column, and each of the two diagonals are the same); instead of picking up a card, the player writes his initials on it. ⦙

10.5 Review

Terms introduced in this chapter:

Redundancy
Parity
Checksum
Check digit
Substitution error
Transposition error
Invariant
Loop invariant
Finite strategy game
Safe and unsafe positions
Strategy stealing
Nim
Nim sum