1 Series 01: variables, expressions and statements

1.1 ISBN

```
# read first nine digits of an ISBN-10 code and convert them to integers
x1 = int(input())
x2 = int(input())
x3 = int(input())
x4 = int(input())
x5 = int(input())
x6 = int(input())
x7 = int(input())
x8 = int(input())
x9 = int(input())

# compute check digit
x10 = (
    x1 + 2 * x2 + 3 * x3 + 4 * x4 + 5 * x5 + 6 * x6 + 7 * x7 + 8 * x8 + 9 * x9
) % 11

# print check digit
print(x10)
```

1.2 Sum of two integers

```
# read two terms and convert them to integers
term1 = int(input(’Give an integer: ’))
term2 = int(input(’Give another integer: ’))

# compute sum of two integers
# NOTE: we avoid using the name "sum" for the variable, as this is the name of
#       a built-in function in Python
total = term1 + term2

# write sum to output
print(total)
```

1.3 Heartbeats

```
# read creature features
creatures = input()          # name of a creature (plural)
heart_rate = int(input())    # heart rate (per minute)
longevity = int(input())     # longevity (in years)

# determine number of heartbeats in a lifetime
minutes_per_year = 60 * 24 * 365
heartbeats = heart_rate * minutes_per_year * longevity

# output number of heartbeats in a lifetime
print(f’{creatures} have {heartbeats / 1e9:.2f} billion heartbeats’)
```

1.4 Timekeeping on Mars

```
# read number of sol
sol = int(input())

# express number of sol in seconds
seconds = int(sol * (((24 * 60) + 39) * 60 + 35.244))
```

```
# convert seconds into minutes and seconds
minutes = seconds // 60
seconds %= 60

# convert minutes into hours and minutes
hours = minutes // 60
minutes %= 60

# convert hours into days and hours
days = hours // 24
hours %= 24

# output conversion of sol into days, hours, minutes and seconds
print(f’{sol} sols = {days} days, {hours} hours, {minutes} minutes and {seconds} seconds’)
```

1.5 The diatomist

```
import math

# read diameter of smaller and larger circles
r = float(input())
R = float(input())

# estimate number of smaller circles that fit into larger circle
count = math.floor(0.83 * (R ** 2 / r ** 2) - 1.9)

# determine coverage of larger circle (percentage)
area_large = math.pi * R ** 2
area_small = math.pi * r ** 2
coverage = (count * area_small) / area_large * 100

# output number of circles and coverage of larger circle
print(f’{count} smaller circles cover {coverage:.2f}% of the larger circle’)
```

1.6 Clock hands

```
# read time on a 24-hour clock
hours = int(input())
minutes = int(input())

# # determine angle that minute hand makes (from 12 o’clock)
# angle_minute = minutes / 60
#
# # determine angle that hour hand makes (from 12 o’clock); take into account that
# # the hour hand also progresses as the minutes pass by
# angle_hour = (hours % 12 + angle_minute) / 12
#
# # determine one of the angles between both hands
# angle_hands = (360 * (angle_hour - angle_minute)) % 360

# some simple arithmetic reduces the above three statements to
angle_hands = (30 * hours - 5.5 * minutes) % 360

# determine smallest angle between both hands
angle_hands = min(angle_hands, 360 - angle_hands)

# output the smallest angle between both hands
print(f’At {hours:02d}:{minutes:02d} both hands form an angle of {angle_hands:.1f}ř.’)
```

2 Series 02: conditional statements

2.1 ISBN

```
# read ten digits of an ISBN-10 code (each on a separate line)
x1 = int(input())
x2 = int(input())
x3 = int(input())
x4 = int(input())
x5 = int(input())
x6 = int(input())
x7 = int(input())
x8 = int(input())
x9 = int(input())
x10 = int(input())

# compute check digit
check_digit = (
    x1 + 2 * x2 + 3 * x3 + 4 * x4 + 5 * x5 + 6 * x6 + 7 * x7 + 8 * x8 + 9 * x9
) % 11

# check and output check digit
print(’OK’ if x10 == check_digit else ’WRONG’)

# alternative solution:
#
# if x10 == check_digit:
#     print(’OK’)
# else:
#     print(’WRONG’)
```

2.2 Personal warmth

```
import math

# read body temperature
body_temperature = float(input())

# make estimate of e
estimate = 100 / body_temperature

# make diagnosis from body temperature
eps = 0.1
if estimate < math.e - eps:
    diagnosis = ’you have a fever’
elif estimate > math.e + eps:
    diagnosis = ’you have hypothermia’
else:
    diagnosis = ’you have a normal body temperature’

# output diagnosis
print(diagnosis)
```

2.3 Mondrian

```
# read coordinate of point on painting
x, y = float(input()), float(input())

# determine color of rectangle that contains the given point
if x < 4.65 and y > 6.0:
    color = ’red’
elif x > 6.3 and y < 2.6:
    color = ’yellow’
elif (2.2 < x < 4.0 and y < 2) or (x > 6.3 and 4.1 < y < 6.0):
    color = ’blue’
else:
    color = ’white’

# print the color
print(color)
```

2.4 Counterfeiting

```
# find group that contains the counterfeit coin based on the first weighing:
# group 0 = 1-2-3; group 1 = 4-5-6; group 2 = 7-8-9
weighing = input()
group = 0 if weighing == ’right’ else (1 if weighing == ’left’ else 2)

# determine which coin in the group is counterfeit
weighing = input()
coin = 1 if weighing == ’right’ else (2 if weighing == ’left’ else 3)

# indicate which coin is counterfeit
print(f’coin #{3 * group + coin} is counterfeit’)
```

2.5 The two towers

```
# read outcome of two coin tosses
# NOTE: outcomes are converted to Boolean values (head -> True, tail -> False) in order to
    simplify the implementation
coin1 = input() == ’heads’
coin2 = input() == ’heads’

# read which scientist will say the same as his own outcome
same = input()

# determine response of both scientists based on the outcome of their own throws and the
    agreement who will say the
# same and who will say the opposite
if same == ’first’:
    coin2 = not coin2
else:
    coin1 = not coin1

# output response of first scientist
print(’heads’ if coin1 else ’tails’)

# output response of second scientist
print(’heads’ if coin2 else ’tails’)
```

2.6 Knight move

```
# read two given position on chess board
position1 = input()
position2 = input()

# decompose positions into row and column indices
col1, row1 = position1
col2, row2 = position2

# convert row indices into integers
row1, row2 = int(row1), int(row2)
```

```
# convert column indices into integers (zero-based)
col1 = ord(col1) - ord(’a’)
col2 = ord(col2) - ord(’a’)

# determine whether a knight can move between two given positions: this is the case if it
    jumps over one column and two
# rows or one row and two columns
conclusion = ’’ if {abs(row1 - row2), abs(col1 - col2)} == {1, 2} else ’not’
print(f’a knight can{conclusion} jump from {position1} to {position2}’)
```

3 Series 03: loops

3.1 ISBN

```
# read first digit of first ISBN-10 code
# NOTE: at this point we cannot assume the first line of the first ISBN-10 code contains a
    digit, since it may also
#       contain the word stop
first_digit = input()

while first_digit != ’stop’:

    # read next eight digits and compute check digit
    computed_check_digit = int(first_digit)
    for index in range(2, 10):
        next_digit = int(input())
        computed_check_digit += index * next_digit
    computed_check_digit %= 11

    # read given check digit
    given_check_digit = int(input())

    # output correctness of given check digit
    print(’OK’ if given_check_digit == computed_check_digit else ’WRONG’)

    # read first digit of next ISBN-10 code
    # NOTE: at this point we cannot assume the first line of the next ISBN-10 code contains a
        digit, since it may also
    #       contain the word stop
    first_digit = input()
```

3.2 Conan the Bacterium

```
# read parameters of experiments
a = int(input())
b = int(input())
n = int(input())
t = int(input())

# determine number of bacteria z obtained after growing a single bacterial strain for n
    seconds
z = 1
for _ in range(n):
    z = a * z + b

# output number of cells obtained after first experiment
print(f’experiment #1: {z} cells after {n} seconds’)

# determine minimal number of seconds needed to grow at least z bacteria starting from t
    bacteria
seconds = 0
while t < z:
    seconds += 1
    t = a * t + b

# output how long cells have to grow during second experiment
print(f’experiment #2: {t} cells after {seconds} seconds’)
```

3.3 Chika’s test

```
# read number
number = int(input())

# first reduction is number itself
# NOTE: we use a new variable that is updated when performing the reduction steps, because we
    still need to print the
#       original number at the end
reduction = number
print(reduction)

# perform reduction steps
while reduction > 9 and reduction != 49:

    # perform reduction step on previous reduction
    reduction = reduction // 10 + 5 * (reduction % 10)

    # output new reduction
    print(reduction)

# output whether number is divisible by 7
seven_test = ’’ if reduction in (7, 49) else ’not ’
print(f’{number} is {seven_test}divisible by 7’)
```

3.4 Challenger or crack

```
# read number of questions in the round
questions = int(input())

# process all answers in the question round
score_challenger, score_crack = 0, 0
for _ in range(questions):

    # process next question: read correct answer and given answers
    correct_answer = input()
    answer_challenger = input()
    answer_crack = input()

    # determine if challenger scores a point on the question
    if answer_challenger == correct_answer:
        score_challenger += 1

    # determine if crack scores a point on the question
    if (answer_crack == ’correct’) == (answer_challenger == correct_answer):
        score_crack += 1

# define a very small value that is used to counter rounding errors when working with floating
     point numbers
eps = 1e-6

# determine outcome of the round
if score_crack < (questions / 2) - eps or score_challenger > score_crack:
    result = f’challenger wins {score_challenger} points against {score_crack}’
elif score_crack == score_challenger:
    result = f’ex aequo: both contestants score {score_crack} points’
else:
    result = f’crack wins {score_crack} points against {score_challenger}’

# output result of the round
print(result)
```

3.5 Payslip

```
# read random number
random_number = int(input())

# initialize total salary with random number
total_salary = random_number

# process salaries of successive workers
salary, workers = input(), 0
while salary != ’stop’:

    # add salary of worker to total salary
    workers += 1
    total_salary += int(salary)

    # output total salary as whispered by worker
    print(f’worker #{workers} whispers {total_salary}’)

    # read salary of next worker
    salary = input()

# output average salary
print(f’average salary: {(total_salary - random_number) / workers:.2f}’)
```

3.6 Heat wave

```
# initialize variables with properties about sequence of successive warm days
summer_days = 0      # number of successive days with temperature above 25 řC
tropical_days = 0    # number of days within sequence of successive days with temperature
    above 30 řC

# no heat wave observed until a sequence of successive days is found that meets the criteria
    of a heat wave
heat_wave = False

# loop over days and determine the length of the current sequence of successive days with a
    temperature above 25 řC, and
# the number of days in that sequence with a temperature above 30 řC
line = input()
while not heat_wave and line != ’stop’:

    # line contains a temperature
    temperature = float(line)

    if temperature >= 25:                   # extend sequence above 25 řC

        summer_days += 1
        if temperature >= 30:
            tropical_days += 1

        # determine if conditions of heat wave have been met
        if summer_days >= 5 and tropical_days >= 3:
            heat_wave = True

    else:                                   # start new sequence above 25 řC

        summer_days = 0
        tropical_days = 0

    # read next line from input
    line = input()

# output whether a heat wave was observed during the given period
print(’heat wave’ if heat_wave else ’no heat wave’)
```

4 Series 04: strings

4.1 ISBN

```
# read first ISBN-10 code (or the word stop)
code = input()

# read successive ISBN-10 codes until line containing "stop" is read
while code != 'stop':

    # compute check digit
    check_digit = int(code[0])
    for i in range(2, 10):
        check_digit += i * int(code[i - 1])
    check_digit %= 11

    # compute check digit: alternative solution using generator expression, slicing and
        enumerate
    # check_digit = sum(index * int(digit) for index, digit in enumerate(code[:-1], start=1))
        % 11

    # check whether computed and extracted check digits are the same
    check = 'OK' if code.endswith('X' if check_digit == 10 else str(check_digit)) else 'WRONG'
    print(check)

    # read next ISBN-10 code (or the word stop)
    code = input()
```

4.2 Babbage’s number

```
# read the number of solutions that must be found
solutions_needed = int(input())

# read the trailing digits of the solution's square
trailing_digits = input()

# output needed number of solutions whose square ends with the given trailing digits
number = -1
solutions_found = 0
while solutions_found < solutions_needed:

    # determine next number
    number += 1

    # check if number is a solution
    if str(number ** 2).endswith(trailing_digits):
        print(f'{number} * {number} = {number ** 2}')
        solutions_found += 1
```

4.3 Mathemagical

```
# read number
number = input()

# determine length of number
length = len(number)

# determine sum from terms consisting of number in which each successive digit is suppressed
total = 0
for index in range(length):

    # determine term by deleting digit at position index in number
    term = int(number[:index] + number[index + 1:])

    # add term to sum
    total += term

    # output term right aligned over length + 1 positions
    # NOTE: put a plus sign before the last term (on the left)
    print(f’{"+" if index == length - 1 else " "}{term:{length}d}’)

# output separator line composed of equal signs
print(’=’ * (length + 1))

# output sum
print(f’{total:{length + 1}d}’)
```

4.4 Reading a pitch

```
# read pitch line
pitch_line = input()

# read starting position and step size
position = int(input())
step = int(input())

# decode pitch line
print(’’.join(
    pitch_line[(position + i * step) % len(pitch_line)] for i in range(len(pitch_line))
))

# # alternative solution:
#
# # decode pitch line
# hidden_message = ’’
# for i in range(len(pitch_line)):
#     hidden_message += pitch_line[(position + i * step) % len(pitch_line)]
#
# # output hidden message
# print(hidden_message)
```

4.5 Reciprocation

```
# read three digit sequences
sequence1 = input()
sequence2 = input()
sequence3 = input()

# process individual digit sequences
for _ in range(3):

    # check which digits in the first sequence correctly indicate the number of
    # occurrences of the corresponding digit in the last two sequences
    correct = ’’
    for index, digit in enumerate(sequence1):
        correct += digit if int(digit) == (sequence2 + sequence3).count(str(index)) else ’X’
    print(correct)

    # shift sequences
    sequence1, sequence2, sequence3 = sequence2, sequence3, sequence1
```

4.6 Word evolutions

```python
from string import ascii_uppercase

# read word on the left
word = input()

# read first letter of word on the right
letter = input()

# repeat alphabet twice; this allows us to cut out the slice that runs from the letter on the
    left up to and including
# the letter on the right
alphabet = 2 * ascii_uppercase

# determine width of slices that need to be cut out by finding the position of the first
    letters of both words, so that
# the position of the first letter of the word on the left comes before the position of the
    first letter of the word on
# the right (hence the need for a double alphabet)
pos1 = alphabet.index(word[0])           # position first letter of left word
pos2 = alphabet.index(letter, pos1 + 1)  # position first letter of right word
width = (pos2 - pos1) + 1

# output evolution from word on the left to word on the right
for letter in word:

    # first position of letter in word on the left in double alphabet
    pos = alphabet.index(letter)

    # cut out slice of fixed width starting at position of letter in the double alphabet
    evolution = alphabet[pos:pos + width]

    # output slice with middle letters converted to lowercase
    print(evolution[0] + evolution[1:-1].lower() + evolution[-1])
```

5 Series 05: functions

5.1 ISBN

```
def isISBN(code):

    """
    Returns True if the argument is a string that represents a valid ISBN-10 code, False
        otherwise.

    >>> isISBN(’9971502100’)
    True
    >>> isISBN(’9971502108’)
    False
    >>> isISBN(’53WKEFF2C’)
    False
    >>> isISBN(4378580136)
    False
    """

    # NOTE: isinstance is a Python built-in function that returns a Boolean value that
        indicates whether the first
    #       argument is an object that has a data type equal to the second argument
    return (
        isinstance(code, str)                # code must be a string,
        and len(code) == 10                  # and code must contain 10 characters,
        and code[:9].isdigit()               # and first nine characters must be digits,
        and checkdigit(code) == code[-1]     # and check digit must be correct
    )

def checkdigit(code):

    """
    Computes the check digit for a given string that contains the first nine digits of an ISBN
        -10 code. A string
    representation of the check digit is returned, with the value 10 represented as the letter
         X.

    >>> checkdigit(’9971502100’)
    ’0’
    >>> checkdigit(’9971502108’)
    ’0’
    """

    # compute check digit
    check = sum(index * int(digit) for index, digit in enumerate(code[:9], start=1)) % 11

    # convert check digit into string representation
    return ’X’ if check == 10 else str(check)

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```

5.2 Noah’s headache

```
def split(species):

    """
    Splits the parameter (str) in a prefix and a suffix, where the prefix is formed by the
        longest sequence of
    consonants at the start of the parameter.

    >>> split(’sheep’)
    (’sh’, ’eep’)
    >>> split('goat')
    ('g', 'oat')
    """

    # find position of first vowel
    pos = 0
    while pos < len(species) and species[pos].lower() not in 'aeiou':
        pos += 1

    # split species name in prefix and suffix
    return species[:pos], species[pos:]

def hybridize(species1, species2):

    """
    Returns a tuple containing two strings. The first element of the tuple is formed by
        concatenating the prefix of the
    first parameter and the suffix of the second parameter. The second element of the tuple is
         formed by concatenating
    the prefix of the second parameter and the suffix of the first parameter.

    >>> hybridize('sheep', 'goat')
    ('shoat', 'geep')
    >>> hybridize('jaguar', 'leopard')
    ('jeopard', 'laguar')
    >>> hybridize('zebra', 'horse')
    ('zorse', 'hebra')
    """

    # split species names in prefix and suffix
    prefix1, suffix1 = split(species1)
    prefix2, suffix2 = split(species2)

    # hybridize the species names
    return prefix1 + suffix2, prefix2 + suffix1

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

5.3 Persistence

```
import math

def multiplication(number):

    """
    >>> multiplication(327)
    42
    >>> multiplication(42)
    8
    >>> multiplication(277777788888899)
    4996238671872
    """

    # multiply all digits of the number together
    return math.prod(int(digit) for digit in str(number))

    # # alternative solution
    # product = 1
    # for digit in str(number):
    #     product *= int(digit)
    # return product

def steps(number):

    """
    >>> steps(327)
    (2, 8)
    >>> steps(68889)
    (7, 0)
    >>> steps(277777788888899)
    (11, 0)
    """

    steps = 0
    while number > 9:
        steps += 1
        number = multiplication(number)

    return steps, number

def digital_root(number):

    """
    >>> digital_root(327)
    8
    >>> digital_root(68889)
    0
    >>> digital_root(277777788888899)
    0
    """

    return steps(number)[1]

def persistence(number):

    """
    >>> persistence(327)
    2
    >>> persistence(8)
    0
    >>> persistence(277777788888899)
    11
    """

    return steps(number)[0]

def most_persistent(lowerbound, upperbound):

    """
    >>> most_persistent(1, 100)
    77
    >>> most_persistent(100, 1000)
    679
    >>> most_persistent(1000, 10000)
    6788
    >>> most_persistent(277777788888000, 277777788889000)
    277777788888899
    """

    return max(range(lowerbound, upperbound + 1), key=persistence)

    # # alternative solution
    # most_persistent_number = lowerbound
    # persistence_number = persistence(lowerbound)
    # for number in range(lowerbound, upperbound):
    #     p = persistence(number + 1)
    #     if p > persistence_number:
    #         persistence_number = p
    #         most_persistent_number = number + 1
    # return most_persistent_number

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```

5.4 Letter Boxed

```
def side(letter, sides):

    """
    >>> side(’O’, ’YOI-RCM-VSA-LTE’)
    0
    >>> side(’V’, ’YOI-RCM-VSA-LTE’)
    2
    >>> side(’E’, ’YOI-RCM-VSA-LTE’)
    3
    >>> side(’X’, ’YOI-RCM-VSA-LTE’)
    -1
    """

    index = sides.find(letter)
    return -1 if index == -1 else sides[:index].count(’-’)

    # alternative solution
    #
    # side = 0
    # for character in sides:
    #     if character == letter:
    #         return side
    #     elif character == ’-’:
    #         side += 1
    #
    # return -1

def iscomplete(chain, sides):

    """
    >>> iscomplete(’MYSTIC-CORAL-LIVER’, ’YOI-RCM-VSA-LTE’)
    True
    >>> iscomplete(’DENIM-MAIZE-EGGPLANT’, ’GND-IET-MZL-AP’)
    True
    >>> iscomplete(’GAINSBORO-ONYX-PEAR’, ’BGY-NXE-PAO-SRI’)
    True
    >>> iscomplete(’KOBI-PLATINUM-ZOMP’, ’TML-OZB-IUK-APN’)
    True
    >>> iscomplete(’DESERT-TEAL-LIVID’, ’SIT-EVW-ADC-KLR’)
    False
    >>> iscomplete(’RUFOUS-SKOBELOFF’, ’FL-XUM-SKE-BOR’)
    False
    >>> iscomplete(’DENIM-OPAL-MANDARIN’, ’NMO-AI-LDR-EYP’)
    False
    >>> iscomplete(’BONE-SEPIA-BROWN’, ’ODI-VAR-BEP-SNW’)
    False
    """

    # all characters must be either a dash or a letter that appearing in at least one word of
        the chain
    return all(character == ’-’ or character in chain for character in sides)

    # alternative solution
    #
    # for character in sides:
    #     if character != ’-’ and character not in chain:
    #         return False
    #
    # return True

def isconsecutive(chain):

    """
    >>> isconsecutive(’MYSTIC-CORAL-LIVER’)
    True
    >>> isconsecutive(’DENIM-MAIZE-EGGPLANT’)
    True
    >>> isconsecutive(’GAINSBORO-ONYX-PEAR’)
    False
    >>> isconsecutive(’KOBI-PLATINUM-ZOMP’)
    False
    >>> isconsecutive(’DESERT-TEAL-LIVID’)
    True
    >>> isconsecutive(’RUFOUS-SKOBELOFF’)
    True
    >>> isconsecutive(’DENIM-OPAL-MANDARIN’)
    False
    >>> isconsecutive(’BONE-SEPIA-BROWN’)
    False
    """

    # determine position of first dash
    index = chain.find(’-’)

    # process all dashes
    while index != -1:

        # check if the same letter appears before and after the dash
        if chain[index - 1] != chain[index + 1]:
            return False

        # determine position of next dash
        index = chain.find(’-’, index + 1)

    return True

    # alternative solution
    #
    # return all(
    #     chain[index - 1] == chain[index + 1]
    #     for index in (index for index, character in enumerate(chain) if character == ’-’)
    # )

def iscrossing(chain, sides):

    """
    >>> iscrossing(’MYSTIC-CORAL-LIVER’, ’YOI-RCM-VSA-LTE’)
    True
    >>> iscrossing(’DENIM-MAIZE-EGGPLANT’, ’GND-IET-MZL-AP’)
    False
    >>> iscrossing(’GAINSBORO-ONYX-PEAR’, ’BGY-NXE-PAO-SRI’)
    True
    >>> iscrossing(’KOBI-PLATINUM-ZOMP’, ’TML-OZB-IUK-APN’)
    False
    >>> iscrossing(’DESERT-TEAL-LIVID’, ’SIT-EVW-ADC-KLR’)
    True
    >>> iscrossing(’RUFOUS-SKOBELOFF’, ’FL-XUM-SKE-BOR’)
    False
    >>> iscrossing(’DENIM-OPAL-MANDARIN’, ’NMO-AI-LDR-EYP’)
    True
    >>> iscrossing(’BONE-SEPIA-BROWN’, ’ODI-VAR-BEP-SNW’)
    False
    """

    # process all successive letter pairs
    for letter1, letter2 in zip(chain, chain[1:]):
        if letter1 != ’-’ and letter2 != ’-’:
            side1 = side(letter1, sides)
            side2 = side(letter2, sides)
            if side1 == -1 or side2 == -1 or side1 == side2:
                return False

    return True

def issolution(chain, sides):
    """
    >>> issolution('MYSTIC-CORAL-LIVER', 'YOI-RCM-VSA-LTE')
    True
    >>> issolution('DENIM-MAIZE-EGGPLANT', 'GND-IET-MZL-AP')
    False
    >>> issolution('GAINSBORO-ONYX-PEAR', 'BGY-NXE-PAO-SRI')
    False
    >>> issolution('KOBI-PLATINUM-ZOMP', 'TML-OZB-IUK-APN')
    False
    >>> issolution('DESERT-TEAL-LIVID', 'SIT-EVW-ADC-KLR')
    False
    >>> issolution('RUFOUS-SKOBELOFF', 'FL-XUM-SKE-BOR')
    False
    >>> issolution('DENIM-OPAL-MANDARIN', 'NMO-AI-LDR-EYP')
    False
    >>> issolution('BONE-SEPIA-BROWN', 'ODI-VAR-BEP-SNW')
    False
    """

    return iscomplete(chain, sides) and isconsecutive(chain) and iscrossing(chain, sides)

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

5.5 Writing's on the wall

```
def digit_sum(number):

    """
    >>> digit_sum(987654321)
    45
    >>> digit_sum(123456789)
    45
    >>> digit_sum(864197532)
    45
    """

    return sum(int(cijfer) for cijfer in str(number))

def issolution(number1, number2):

    """
    >>> issolution(987654321, 123456789)
    True
    >>> issolution(93243, 58134)
    False
    """

    first_sum = digit_sum(number1)
    return first_sum == digit_sum(number2) and first_sum == digit_sum(number1 - number2)

def solutions(lower_limit, upper_limit):

    """
    >>> solutions(1000, 2000)
    1311
    >>> solutions(2000, 3000)
    1202
    >>> solutions(3000, 4000)
    1128
    """

    count = 0
    for number1 in range(lower_limit + 1, upper_limit + 1):
        for number2 in range(lower_limit, number1):
            if issolution(number1, number2):
                count += 1

    return count

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

5.6 Tap code

```
"""
>>> encode_letter('V')
(5, 1)
>>> encode_letter('i')
(2, 4)
>>> encode('VICTOR')
'..... . .. .... . ... .... .... ... .... .... ..'
>>> encode('Charlie')
'. ... .. ... . . .... .. ... . .. .... . .....'

>>> decode_letter(5, 1)
'V'
>>> decode_letter(2, 4)
'I'
>>> decode('..... . .. .... . ... .... .... ... .... .... ..')
'VICTOR'
>>> decode('. ... .. ... . . .... .. ... . .. .... . .....')
'CHARLIE'
"""

import string

alphabet = string.ascii_uppercase.replace('K', '')

def encode_letter(letter):

    letter = letter.upper().replace('K', 'C')
    index = alphabet.index(letter)
    return index // 5 + 1, index % 5 + 1

def encode(word):

    lengths = []
    for letter in word:
        lengths.extend(encode_letter(letter))

    return ' '.join(length * '.' for length in lengths)

def decode_letter(row, col):

    return alphabet[(row - 1) * 5 + (col - 1)]

def decode(taps):

    word = ''
    first = None
    for tap in [len(tap) for tap in taps.split()]:
        if first is None:
            first = tap
        else:
            word += decode_letter(first, tap)
            first = None

    return word

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

6 Series 06: lists and tuples

6.1 ISBN

```
def isISBN(code):

    """
    Checks if an ISBN-10 code is valid.

    >>> isISBN(’9-9715-0210-0’)
    True
    >>> isISBN(’997-150-210-0’)
    False
    >>> isISBN(’9-9715-0210-8’)
    False
    """

    # check if the code is a string
    if not isinstance(code, str):
        return False

    # check if dashes are at correct positions and if each group has correct number of digits
    groups = code.split(’-’)
    if [len(e) for e in groups] != [1, 4, 4, 1]:
        return False

    # remove dashes from the code
    code = ’’.join(groups)

    # check if all characters (except the final one) are digits
    if not code[:-1].isdigit():
        return False

    # check the check digit of the code
    return check_digit(code) == code[-1]

def check_digit(code):

    """
    >>> check_digit(’997150210’)
    ’0’
    >>> check_digit(’938389293’)
    ’5’
    """

    # compute check digit
    check = sum(index * int(digit) for index, digit in enumerate(code[:9], start=1)) % 11

    # convert check digit into its string representation
    return ’X’ if check == 10 else str(check)

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```

7 Series 07: more about functions and modules

7.1 ISBN

```
def isISBN10(code):

    """
    Checks whether the ISBN-10 code is valid.

    >>> isISBN10(’9971502100’)
    True
    >>> isISBN10(’9971502108’)
    False
    """

    # helper function for computing ISBN-10 check digit
    def check_digit(code):

        # compute check digit
        check = sum(index * int(digit) for index, digit in enumerate(code[:9], start=1)) % 11

        # convert check digit into its string representation
        return ’X’ if check == 10 else str(check)

    # check whether the code is a string
    if not isinstance(code, str):
        return False

    # check whether the code contains 10 characters
    if len(code) != 10:
        return False

    # check whether first nine characters of the code are digits
    if not code[:9].isdigit():
        return False

    # check the check digit
    return check_digit(code) == code[-1]

def isISBN13(code):

    """
    Checks whether the ISBN-13 code is valid.

    >>> isISBN13(’9789743159664’)
    True
    >>> isISBN13(’9787954527409’)
    False
    >>> isISBN13(’8799743159665’)
    False
    """

    # helper function for computing ISBN-10 check digit
    def check_digit(code):

        # compute check digit
        check = sum((3 if index % 2 else 1) * int(digit) for index, digit in enumerate(code
            [:12]))

        # convert check digit into a single digit
        return str((10 - check) % 10)

    # check whether the code is a string
    if not isinstance(code, str):
        return False

    # check whether the code contains 10 characters
    if len(code) != 13:
        return False

    # check whether first nine characters of the code are digits
    if not code[:12].isdigit():
        return False

    # check the check digit
    return check_digit(code) == code[-1]

def isISBN(code, isbn13=True):

    """
    >>> isISBN(’9789027439642’, False)
    False
    >>> isISBN(’9789027439642’, True)
    True
    >>> isISBN(’9789027439642’)
    True
    >>> isISBN(’080442957X’)
    False
    >>> isISBN(’080442957X’, False)
    True
    """

    return isISBN13(code) if isbn13 else isISBN10(code)

def areISBN(codes, isbn13=None):

    """
    >>> codes = [’0012345678’, ’0012345679’, ’9971502100’, ’080442957X’, 5, True, ’The
        Practice of Computing Using Python’, ’9789027439642’, ’5486948320146’]
    >>> areISBN(codes)
    [False, True, True, True, False, False, False, True, False]
    >>> areISBN(codes, True)
    [False, False, False, False, False, False, False, True, False]
    >>> areISBN(codes, False)
    [False, True, True, True, False, False, False, False, False]
    """

    # initialize list of checks
    checks = []

    # construct list of checks
    for code in codes:
        checks.append(
            False if not isinstance(code, str)
            else isISBN(code, len(code) == 13 if isbn13 is None else isbn13)
        )

    # return list of checks
    return checks

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```

8 Series 08: sets and dictionaries

8.1 ISBN

```
def isISBN13(code):

    """
    Checks whether the ISBN-13 code is valid.

    >>> isISBN13(’9789743159664’)
    True
    >>> isISBN13(’9787954527409’)
    False
    >>> isISBN13(’8799743159665’)
    False
    """

    def check_digit(code):

        """
        Helper function that computes the ISBN-13 check digit.
        """

        # compute check digit
        check = sum(
            (3 if index % 2 else 1) * int(digit)
            for index, digit in enumerate(code[:12])
        )

        # convert check digit into a single digit
        return str((10 - check) % 10)

    # check if the code is a string
    if not isinstance(code, str):
        return False

    # check if the code contains 13 characters
    if len(code) != 13:
        return False

    # check prefix of the code
    if code[:3] not in {’978’, ’979’}:
        return False

    # check if all characters of the code are digits
    if not code.isdigit():
        return False

    # check the check digit
    return check_digit(code) == code[-1]

def overview(codes):

    """
    >>> codes = [
    ...     ’9789743159664’, ’9785301556616’, ’9797668174969’, ’9781787559554’,
    ...     ’9780817481461’, ’9785130738708’, ’9798810365062’, ’9795345206033’,
    ...     ’9792361848797’, ’9785197570819’, ’9786922535370’, ’9791978044523’,
    ...     ’9796357284378’, ’9792982208529’, ’9793509549576’, ’9787954527409’,
    ...     ’9797566046955’, ’9785239955499’, ’9787769276051’, ’9789910855708’,
    ...     ’9783807934891’, ’9788337967876’, ’9786509441823’, ’9795400240705’,
    ...     ’9787509152157’, ’9791478081103’, ’9780488170969’, ’9795755809220’,
    ...     ’9793546666847’, ’9792322242176’, ’9782582638543’, ’9795919445653’,
    ...     ’9796783939729’, ’9782384928398’, ’9787590220100’, ’9797422143460’,
    ...     ’9798853923096’, ’9784177414990’, ’9799562126426’, ’9794732912038’,
    ...     ’9787184435972’, ’9794455619207’, ’9794270312172’, ’9783811648340’,
    ...     ’9799376073039’, ’9798552650309’, ’9798485624965’, ’9780734764010’,
    ...     ’9783635963865’, ’9783246924279’, ’9797449285853’, ’9781631746260’,
    ...    ’9791853742292’, ’9781796458336’, ’9791260591924’, ’9789367398012’
    ... ]
    >>> overview(codes)
    English speaking countries: 8
    French speaking countries: 4
    German speaking countries: 6
    Japan: 3
    Russian speaking countries: 7
    China: 8
    Other countries: 11
    Errors: 9
    """

    # construct histogram of registration groups
    groups = {group: 0 for group in range(11)}
    for code in codes:
        group = int(code[3]) if isISBN13(code) else 10
        groups[group] += 1

    # display overview
    print(f’English speaking countries: {groups[0] + groups[1]}’)
    print(f’French speaking countries: {groups[2]}’)
    print(f’German speaking countries: {groups[3]}’)
    print(f’Japan: {groups[4]}’)
    print(f’Russian speaking countries: {groups[5]}’)
    print(f’China: {groups[7]}’)
    print(f’Other countries: {groups[6] + groups[8] + groups[9]}’)
    print(f’Errors: {groups[10]}’)

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```

9 Series 09: text files

9.1 ISBN

```
import urllib.request

def isISBN13(code):

    """
    Returns a Boolean value that indicates whether the given ISBN-13 code is valid.

    >>> isISBN13(’9789743159664’)
    True
    >>> isISBN13(’9787954527409’)
    False
    >>> isISBN13(’8799743159665’)
    False
    """

    def checkdigit(code):

        """
        Helper function that computes the ISBN-13 check digit.
        """

        # compute check digit
        check = sum(
            (3 if index % 2 else 1) * int(digit)
            for index, digit in enumerate(code[:12])
        )

        # convert check digit into a single digit
        return str((10 - check) % 10)

    # check whether the code is a string
    if not isinstance(code, str):
        return False

    # check whether the code contains 13 characters
    if len(code) != 13:
        return False

    # check prefix of the code
    if code[:3] not in {’978’, ’979’}:
        return False

    # check whether first nine characters of the code are digits
    if not code[:12].isdigit():
        return False

    # check the check digit
    return checkdigit(code) == code[-1]

def remove_tags(s):

    """
    Removes all XML tags from the given string and then removes all leading and trailing
        whitespace.

    >>> remove_tags(’ <Title> The Practice of Computing using <b>Python</b> </Title> ’)
    ’The Practice of Computing using Python’
    """

    # remove all XML tags from the string
    start = s.find(’<’)
    while start >= 0:
        stop = s.find(’>’, start + 1)
        if stop == -1:
            stop = len(s)
        s = s[:start] + s[stop + 1:]
        start = s.find(’<’)

    # remove leading and trailing whitespace and returns the modified string
    return s.strip()

def display_book_info(code):

    """
    >>> display_book_info(’9780136110675’)
    Title: The Practice of Computing using Python
    Authors: William F Punch, Richard Enbody
    Publisher: Addison Wesley
    >>> display_book_info(’9780136110678’)
    Wrong ISBN-13 code
    """

    # remove leading and trailing whitespace characters from code
    code = code.strip()

    # check validity of ISBN-13 code
    if not isISBN13(code):

        # print error message in case given ISBN-13 code is invalid
        print(’Wrong ISBN-13 code’)

    else:

        # construct URL of imitated ISBNdb.com that provides information about the book with
            the ISBN-13 code
        url = f’https://pythia.ugent.be/pythia-share/exercises/isbn9/books.php?isbn={code}’

        # extract and output selected book information from XML
        with urllib.request.urlopen(url) as info:
            for line in info:
                line = line.decode(’utf-8’)
                if line.startswith(’<Title>’):
                    print(f’Title: {remove_tags(line)}’)
                elif line.startswith(’<AuthorsText>’):
                    print(f’Authors: {remove_tags(line).rstrip(", ")}’)
                elif line.startswith(’<PublisherText ’):
                    print(f’Publisher: {remove_tags(line).rstrip(", ")}’)

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```

10 Series 10: object-oriented programming

10.1 ISBN

```
class ISBN13:

    """
    >>> code = ISBN13(9780136110675)
    >>> print(code)
    978-0-13611067-5
    >>> code
    ISBN13(9780136110675, 1)
    >>> code.isvalid()
    True
    >>> code.asISBN10()
    ’0-13611067-3’

    >>> ISBN13(9780136110675, 6)
    Traceback (most recent call last):
    AssertionError: invalid ISBN code
    """

    def __init__(self, code, length=1):

        # check validity of arguments
        assert (
            isinstance(code, int)        # code is an integer
            and isinstance(length, int)  # length is an integer
            and 1 <= length <= 5         # length is in the interval [1, 5]
        ), ’invalid ISBN code’

        # object properties: ISBN-code and length of country group convert to string of 13
            characters with leading zeros
        self.code = f’{code:013d}’
        self.length = length

    def __int__(self):

        return int(self.code)

    def __str__(self):

        # return formatted representation of ISBN-code
        c = self.code
        return f’{c[:3]}-{c[3:3 + self.length]}-{c[3 + self.length:-1]}-{c[-1]}’

    def __repr__(self):

        # return string containing a Python expression that creates a new object having the
            same internal state as the
        # current object
        return f’ISBN13({int(self.code)}, {self.length})’

    def isvalid(self):

        def checkdigit(code):

            # compute ISBN-13 check digit
            check = sum(
                (3 if index % 2 else 1) * int(digit)
                for index, digit in enumerate(code[:12])
            )

            # convert check digit into string representation
            return str((10 - check) % 10)

        # check validity of check digit
        return self.code[12] == checkdigit(self.code)
    def asISBN10(self):

        def checkdigit(code):

            # compute ISBN-10 check digit
            check = sum(
                index * int(digit)
                for index, digit in enumerate(code[:9], start=1)
            ) % 11

            # convert check digit into string representation
            return ’X’ if check == 10 else str(check)

        # return no result for invalid ISBN-13 codes
        if not self.isvalid() or not self.code.startswith(’978’):
            return None

        # convert ISBN-13 code into ISBN-10 code
        code = self.code[3:-1]
        check = checkdigit(code)
        return f’{code[:self.length]}-{code[self.length:]}-{check}’

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```