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 Best laid plans

```
# read data
A = int(input())     # mistakes found by first examiner
B = int(input())     # mistakes found by second examiner
C = int(input())     # common mistakes found

# estimate total number of undiscovered mistakes
undiscovered_errors = (A - C) * (B - C) / C

# output number of undiscovered mistakes
print(f'There are {undiscovered_errors:.2f} undiscovered errors.')
```

1.4 The pudding guy

```
# read problem parameters
items = int(input())        # total number of items purchased
price = float(input())      # cost per item
UPCs = int(input())         # number of UPCs in a series
miles = int(input())        # number of miles earned per series of UPCs

# compute total amount of money spent
```

```
spent = items * price

# compute total number of frequent flyer miles earned
earned = (items // UPCs) * miles

# generate formatted output
print(f'Phillips spent ${spent} for {earned} frequent-flyer miles.')
```

1.5 Human Development Index

```
# add extra mathematical functionality to Python
#
#   sqrt: square root
#   log: (natural) logarithm
#
# note: the math module is part of The Python Standard Library
from math import sqrt, log

# read parameters that are used for computing the Human Development Index
country = input()       # name of the country
LE = float(input())     # life expectancy at birth
MYS = float(input())    # mean years of schooling for a 25-year-old
EYS = float(input())    # expected years of schooling for a 5-year-old
GNI = float(input())    # gross national income per capita

# determine three normalized indices that are used in computing the Human Development Index
LEI = (LE - 20.0) / (82.3 - 20.0)     # Life Expectancy Index
MYSI = MYS / 13.2                     # Mean Years of Schooling Index
EYSI = EYS / 20.6                     # Expected Years of Schooling Index
EI = sqrt(MYSI * EYSI) / 0.951        # Education Index
II = (log(GNI) - log(100.0)) / (log(107721) - log(100))

# compute Human Development Index as the geometric mean of the previous three normalized
    indices
HDI = pow(LEI * EI * II, 1 / 3)

# print value of the Human Development Index; format specifier .3f is used to output a
    floating point number with three
# decimal digits (also uses rounding when truncating the floating point number)
print(f'The HDI of {country} is {HDI:.3f}.')
```

1.6 The stopped clock

```
# read time on four successive 24-hour clocks (hh:mm)
h1, m1 = int(input()), int(input())
h2, m2 = int(input()), int(input())
h3, m3 = int(input()), int(input())
h4, m4 = int(input()), int(input())

# number of minutes per hour and per day
minutes_per_hour = 60
minutes_per_day = 24 * minutes_per_hour

# convert time on all 24-hour clocks to number of minutes since midnight
m1 += h1 * minutes_per_hour
m2 += h2 * minutes_per_hour
m3 += h3 * minutes_per_hour
m4 += h4 * minutes_per_hour

# compute correction between stopped clock and correct clock
total_time = (m4 - m1) % minutes_per_day        # read from stopped clock
visit_time = (m3 - m2) % minutes_per_day        # read from correct clock
travel_time = (total_time - visit_time) // 2    # one way travel time
```

```
# compute correct time upon arrival at home
# NOTE: modulo operation assures that the time is always expressed as a number of minutes
    since midnight
corrected_time = (m3 + travel_time) % minutes_per_day

# convert corrected time in hours and minutes
print(corrected_time // minutes_per_hour)
print(corrected_time % minutes_per_hour)
```

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 Reynolds number

```
# read parameters
velocity = float(input())
length = float(input())
density = float(input())
viscosity = float(input())

# calculate Reynolds number
reynolds_number = (velocity * length * density) / viscosity

# determine type of flow
if reynolds_number < 2000:
    type_of_flow = 'laminar flow'
elif reynolds_number <= 4000:
    type_of_flow = 'transition flow'
else:
    type_of_flow = 'turbulent flow'

# output result
print(f'{reynolds_number} ({type_of_flow})')
```

2.3 Rock-paper-scissors

```
# read hand signs of both players
player1 = input()
player2 = input()

# determine outcome of rock-paper-scissors game
if player1 == player2:
    outcome = 'tie'
```

```
elif (
    (player1 == 'paper' and player2 == 'rock')
    or (player1 == 'rock' and player2 == 'scissors')
    or (player1 == 'scissors' and player2 == 'paper')
):
    outcome = 'player1 wins'
else:
    outcome = 'player2 wins'

# print outcome of rock-paper-scissors game
print(outcome)

# alternative solution (uses a tuple)
#
# # define sequence of possible hand signs in such a way that each element in the sequence
    wins agains the next element
# # in the sequence (where the last element in the sequence is followed by the first element
    in the sequence)
# hand_signs = ('paper', 'rock', 'scissors')
#
# # read hand signs of both players and convert them to their position in the sequence of
    possible hand signs
# player1 = hand_signs.index(input())
# player2 = hand_signs.index(input())
#
# # determine outcome of rock-paper-scissors game
# outcomes = ('tie', 'player1 wins', 'player2 wins')
# outcome = outcomes[(player2 - player1) % 3]
#
# # print outcome of rock-paper-scissors game
# print(outcome)


# alternative solution (uses a dictionary)
#
# # create dictionary that defines winners (key) and losers (values)
# wins_against = {'paper': 'rock', 'rock': 'scissors', 'scissors': 'paper'}
#
# # determine outcome of rock-paper-scissors game
# if player1 == player2:
#     outcome = 'tie'
# elif wins_against[player1] == player2:
#     outcome = 'player1 wins'
# else:
#     outcome = 'player2 wins'
#
# # print outcome of rock-paper-scissors game
# print(outcome)
```

2.4 Seasons

```
# read day and month of the given date
day = int(input())
month = input()

# convert name of month to index of the month in the year
months = [
    'January', 'February', 'March', 'April', 'May', 'June', 'July',
    'August',  'September', 'October', 'November', 'December'
]
index = months.index(month)

# determine season for given day in the year
if index < 2 or (index == 2 and day < 21):
    season = 'winter'
elif index < 5 or (index == 5 and day < 21):
    season = 'spring'
```

```
elif index < 8 or (index == 8 and day < 23):
    season = 'summer'
elif index < 11 or (index == 11 and day < 21):
    season = 'autumn'
else:
    season = 'winter'

# alternative solution (using tuples)
#
# if (index, day) < (2, 21) or (index, day) > (11, 20):
#     season = 'winter'
# elif (index, day) < (5, 21):
#     season = 'spring'
# elif (index, day) < (8, 23):
#     season = 'summer'
# else:
#     season = 'autumn'

# print corresponding season
print(f'It is {season} on {month} {day}.')
```

2.5 Trilateration

```
# read coordinates of first circle
x1 = float(input())
y1 = float(input())
r1 = float(input())

# read coordinates of second circle
x2 = float(input())
y2 = float(input())
r2 = float(input())

# determine Euclidean distance between two circle centers
distance = ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

# define margin of error when comparing floating point numbers
eps = 1e-6

# determine relative position of two given circles
if distance < eps:
    position = 'concentric'
elif abs(distance - abs(r1 - r2)) < eps:
    position = 'touching inside'
elif abs(distance - (r1 + r2)) < eps:
    position = 'touching outside'
elif distance < abs(r1 - r2):
    position = 'enclosing'
elif distance < r1 + r2:
    position = 'intersecting'
else:
    position = 'separate'

# output relative position of two given circles
print(position)
```

2.6 Darts

```
# import extra math functions
from math import sqrt, atan2, pi

# read position in Cartesian coordinates
x, y = float(input()), float(input())
```

```
# convert Cartesian coordinates into polar coordinates
eps = 1e-6
if abs(x) < eps and abs(y) < eps:
    radius, angle = 0.0, 0.0
else:
    radius, angle = sqrt(x ** 2 + y ** 2), atan2(y, x)

# some scores only depend on the radius r; in these cases there is not need to determine the
    index of the sector in
# which the dart is thrown
if radius >= 170.0:

    # darts outside the board get no score
    score = 0

elif radius < 12.7 / 2:

    # bull's eye
    score = 50

elif radius < 31.8 / 2:

    # bull
    score = 25

else:

    # invert and shift angle in polar coordinates
    # NOTE: modulo because atan2 function returns angle in interval [-, [
    sectors = 20
    angle = (pi / 2.0 + pi / sectors - angle) % (2 * pi)

    # compute index of sector
    index = int(angle * sectors / (2 * pi))

    # basic score is one unit higher than index
    score = index + 1

    # correct score based on distance to center
    if 107.0 - 9.6 < radius < 107.0:
        # triple ring
        score *= 3
    elif 170.0 - 9.6 < radius < 170.0:
        # double ring
        score *= 2

# print the final score
print(score)
```

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 along the way
    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 Chaos

```
# read simulation parameters
density = float(input())
fecundicity = float(input())
steps = int(input())

# output initial density
print(density)

# output density during successive simulation steps
for _ in range(steps - 1):

    # compute next density
    density = fecundicity * density * (1.0 - density)

    # output next density
    print(density)
```

3.3 German tanks

```
# initialize count and maximum serial number
count, maximum = 0, 0

# read first serial number
serial_number = int(input())

# process serial numbers until negative number is found
while serial_number >= 0:

    # count numbers and determine maximal number while reading the numbers
```

```
    count += 1
    maximum = max(maximum, serial_number)

    # read next serial number
    serial_number = int(input())

# estimate number of produced tanks
tanks = ((count + 1) * maximum) / count - 1

# output estimate
print(f'The number of produced tanks is estimated to be {round(tanks)}.')
```

3.4 Pythagorean triples

```
import math

# read sum of triples
n = int(input())

# evaluate all possible triples, but skip triples that can never meet the conditions as stated
    in the assignment; we
# have for a that 1  a (lower limit) and also that
#
#     a  b  c
#     a + b + c = n
#
# from which it follows that
#
#     a + a + a  n
#
# or (upper limit)
#
#     a  n / 3
#
for a in range(1, math.ceil(n / 3) + 1):

    # if a is fixed, we have for b that a  b (lower limit) and also that
    #
    #     b  c
    #     b + c = n - a
    #
    # from which it follows that
    #
    #     b + b  n - a
    #
    # or (upper limit)
    #
    #     b  (n - a) / 2
    #
    for b in range(a, math.ceil((n - a) / 2) + 1):

        # if a and b are fixed, then c is also fixed because a + b + c = n
        c = n - a - b

        # only output triples if the numbers can be the integer side lengths of a right
            triangle
        if a ** 2 + b ** 2 == c ** 2:
            print(f'({a}, {b}, {c})')
```

3.5 Monkeys and coconuts

```
def count(nuts):

    """
```

```
    Returns a description of a number of coconuts, using correct singular or plural form in
        the description.
    """

    return f'{str(nuts) if nuts else 'no'} {'nut' if nuts == 1 else 'nuts'}'

# read the number of pirates
pirates = int(input())

# read the number of coconuts in the pile at the start
coconuts = int(input())

# at night each of pirate takes its share and gives remaining ones to monkey
template = '{} = {} for pirate#{} and {} for the monkey'
for pirate in range(1, pirates + 1):

    # compute number of coconuts pirate gives to himself and the monkey
    coconuts_for_pirate = coconuts // pirates
    coconuts_for_monkey = coconuts % pirates

    # output number of coconuts pirate gives to himself and the monkey
    print(template.format(
        count(coconuts),
        count(coconuts_for_pirate),
        pirate,
        count(coconuts_for_monkey)
    ))

    # remove coconuts for pirate and monkey from the pile of coconuts
    coconuts -= coconuts_for_pirate + coconuts_for_monkey

# determine share of coconuts for each pirate and monkey when morning breaks
print(f'each pirate gets {count(coconuts // pirates)} and {count(coconuts % pirates)} for the
    monkey')
```

3.6 Hitchhikers problem

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

# determine maximal score among first half of the hitchhikers
maximal_score = 0
half = hitchhikers // 2
for _ in range(half):
    maximal_score = max(maximal_score, float(input()))

# judge next hitchhiker
hitchhiker = half + 1
score = float(input())

# continue while more hitchhikers follow and current hitchhiker scores lower than hitchhikers
    in first half
while hitchhiker < hitchhikers and score < maximal_score:
    hitchhiker += 1
    score = float(input())

# output score of hitchhiker that is selected
print(score)
```

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

    # convert check digit to its string representation
    check_digit = 'X' if check_digit == 10 else str(check_digit)

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

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

4.2 Journal to Stella

```
# read encoded message
ciphertext = input()

# decode message by discarding alternate letters
# NOTE: characters that are no letters must stay in place
plaintext = ''
keep = True
for character in ciphertext:
    if character.isalpha():
        if keep:
            plaintext += character
        keep = not keep
    else:
        plaintext += character

# output message
print(plaintext)
```

4.3 A needle in a haystack

```
# read word to find and convert it to uppercase
word = input()
WORD = word.upper()

# read number of rows in the grid
rows = int(input())

# search grid row by row; stop when all rows have been processed or when word has been found
row_index = -1
```

```
col_index = -1
while row_index < rows - 1 and col_index == -1:

    # read next line and convert to uppercase
    row = input().upper()
    row_index += 1

    # find position of word in row
    col_index = row.find(WORD)

    # find position of reverse word in row
    if col_index == -1:
        col_index = row.find(WORD[::-1])

# output position where word is found in the row
if col_index == -1:
    print(f'{word} is not found')
else:
    print(f'{word} is on row {row_index} and column {col_index}')
```

4.4 Easy does it

```
# read key and ciphertext
key = input()
ciphertext = input()

# compute plaintext
plaintext = ''
for digit1 in map(str, range(10)):
    for digit2, character in zip(key, ciphertext):
        if digit1 == digit2:
            plaintext += character

# output the ciphertext
print(plaintext)
```

4.5 The letters of Utrecht

```
# read sentence
sentence = input()

# read where we are looking to count letters: past (back), present (around) or future (ahead)
looking = input()

# convert sentence into lowercase
# NOTE: to count letters without making a distinction between uppercase and lowercase
sentence_lowercase = sentence.lower()

# align letters with their number of occurrences
aligned_sentence = ''
aligned_counts = ''
for index, character in enumerate(sentence):
    if character.isalpha():
        count = str(sentence_lowercase.count(
            character.lower(),
            # count from next position onwards if looking to the future, otherwise from the
                start
            index + 1 if looking == 'future' else 0,
            # count up to the current position if looking into the past, otherwise all the way
                 to the end
            index if looking == 'past' else len(sentence)
        ))
        aligned_sentence += f'{character:-<{len(count)}s}'
        aligned_counts += count
```

```
    else:
        aligned_sentence += character
        aligned_counts += character

# show letter occurrences aligned against letters in the sentence
print(aligned_sentence)
print(aligned_counts)
```

4.6 Wepe speapeak p

```
# read sentence in language P
sentence = input()

# convert sentence from p-language to plain English
index = 0
translation = ''
vowel_combination = ''
while index < len(sentence):

    # extract letter at the current position (index)
    letter = sentence[index]

    if letter.lower() == 'p' and vowel_combination:

        # letter p preceded by vowel combination indicates that following
        # vowel combination (having the same length as the preceding vowel
        # combination) must be skipped; the letter p must be skipped as well
        index += len(vowel_combination)
        vowel_combination = ''

    else:

        # add letter to translation
        translation += letter

        # remember vowel combination
        if letter.lower() in 'aeiou' or (vowel_combination + letter).lower() == 'ij':
            vowel_combination += letter
        else:
            vowel_combination = ''

    # go to the next positions
    index += 1

# print plain English translation of the given sentence
print(translation)

# alternative solution
#
# # import module that allows us to work with regular expressions
# import re
#
# # read sentence in language P
# sentence = input()
#
# # translation van de sentence uitschrijven
# print(re.sub(r'(ij|[aeiou]+)p\1', r'\1', sentence, flags=re.IGNORECASE))
```

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 Word sums

```
def letter_value(letter):

    """
    >>> letter_value('A')
    1
    >>> letter_value('j')
    10
    >>> letter_value('!')
    0
```

```
    """

    # letter value = position of given letter in alphabet
    return ord(letter.upper()) - ord('A') + 1 if letter.isalpha() else 0

def word_value(word):

    """
    >>> word_value('arm')
    32
    >>> word_value('BEND')
    25
    >>> word_value('elbow')
    57
    >>> word_value('Ghent')
    """

    # compute sum of letter values of all letters in the word
    return sum(letter_value(letter) for letter in word)

def is_word_sum(word1, word2, word3):

    """
    >>> is_word_sum('arm', 'BEND', 'elbow')
    True
    >>> is_word_sum('KING', 'chair', 'THRONE')
    True
    >>> is_word_sum('Monty', 'Python', 'SHRUBBERY')
    False
    """

    # sum of word values of first two words =?= word value of third word
    return word_value(word1) + word_value(word2) == word_value(word3)

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

5.3 The last banana

```
def has_won(roll1, roll2, limit):

    """
    >>> has_won(3, 2, 4)
    True
    >>> has_won(1, 5, 4)
    False
    """

    return max(roll1, roll2) <= limit

def winning_outcomes(sides, limit):

    """
    >>> winning_outcomes(6, 4)
    (16, 20)
    """

    player1 = limit * limit
    player2 = sides * sides - player1
    return player1, player2

    # player1, player2 = 0, 0
    # for roll1 in range(sides):
    #     for roll2 in range(sides):
    #         if has_won(roll1 + 1, roll2 + 1, limit):
    #             player1 += 1
```

```
    #         else:
    #             player2 += 1
    #
    # return player1, player2

def odds(sides, limit):

    """
    >>> odds(6, 4)
    (44.44444444444444, 55.55555555555556)
    """

    player1, player2 = winning_outcomes(sides, limit)
    total = player1 + player2
    return (player1 / total) * 100, (player2 / total) * 100

def winner(sides, limit):

    """
    >>> winner(6, 4)
    2
    """

    player1, player2 = winning_outcomes(sides, limit)
    return 2 if player2 > player1 else (1 if player1 > player2 else 0)

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

5.4 Rearview mirror

```
def letter2position(letter):

    """
    >>> letter2position('a')
    0
    >>> letter2position('Z')
    25
    """

    return ord(letter.lower()) - ord('a')

def position2letter(position):

    """
    >>> position2letter(0)
    'a'
    >>> position2letter(25)
    'z'
    """

    return chr(ord('a') + position % 26)

def encode_letter(letter, previous):

    """
    >>> encode_letter('d', 'Z')
    'c'
    >>> encode_letter('T', 'e')
    'X'
    """

    ciphertext = position2letter(letter2position(letter) + letter2position(previous))
    return ciphertext.upper() if letter.isupper() else ciphertext

def decode_letter(letter, previous):
```

```
    """
    >>> decode_letter('c', 'Z')
    'd'
    >>> decode_letter('X', 'e')
    'T'
    """

    plaintext = position2letter(letter2position(letter) - letter2position(previous))
    return plaintext.upper() if letter.isupper() else plaintext

def encode(message):

    """
    >>> encode('Saw things clearer once you were in my rear-view mirror.')
    'Ssw papvty unpervv fbpg cmi qavv mv zk pver-mdma iuziff.'
    >>> encode('The past is always tense. The future perfect.')
    'Tal tpsl ba slhwyq lxrfw. Xal jznnlv ttvwjgv.'
    """

    ciphertext = ''
    previous = None
    for character in message:
        if character.isalpha():
            ciphertext += encode_letter(character, previous) if previous else character
            previous = character
        else:
            ciphertext += character

    return ciphertext

def decode(message):

    """
    >>> decode('Ssw papvty unpervv fbpg cmi qavv mv zk pver-mdma iuziff.')
    'Saw things clearer once you were in my rear-view mirror.'
    >>> decode('Tal tpsl ba slhwyq lxrfw. Xal jznnlv ttvwjgv.')
    'The past is always tense. The future perfect.'
    """

    plaintext = ''
    previous = None
    for character in message:
        if character.isalpha():
            previous = decode_letter(character, previous) if previous else character
            plaintext += previous
        else:
            plaintext += character

    return plaintext

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

5.5 Inventory

```
import string

def inventory(sentence):

    """
    >>> inventory('Data aequatione quotcunque fluentes quantitates involvente, fluxiones
        invenire; et vice versa.')
    '7accd14eff7i3l9n4o4qrr4s9t7u5vx'
    >>> inventory('ut tensio sic vis')
    'ce3ino3sttuv'
```

```
    """

    # convert all letters into lowercase
    sentence = sentence.lower()

    # add encoding for each letter in the alphabet (that occurs in the sentence)
    encoding = ''
    for letter in string.ascii_lowercase:
        # count occurrences of letter in sentence
        count = sentence.count(letter)
        # determine letter encoding based on number of occurrences
        encoding += f'{count}{letter}' if count > 2 else letter * count

    return encoding

def unpack(sentence):

    """
    >>> unpack('6accdae13eff7i3l9n4o4qrr4s8t12ux')
    'aaaaaaccdaeeeeeeeeeeeeeeffiiiiiiiilllnnnnnnnnnooooqqqqrrsssstttttttttuuuuuuuuuuuux'
    >>> unpack('ce3ino3sttuv')
    'ceiiinossttuv'
    """

    number = ''
    result = ''
    for character in sentence:
        if character.isdigit():
            number += character
        elif number:
            result += character * int(number)
            number = ''
        else:
            result += character

    return result

def simplify(sentence):

    """
    >>> simplify('6accdae13eff7i3l9n4o4qrr4s8t12ux')
    '7accd14eff7i3l9n4o4qrr4s8t12ux'
    >>> simplify('i2scvotu2ietns')
    'ce3ino3sttuv'
    """

    return inventory(unpack(sentence))

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

5.6 Blinkenlights

```
"""
>>> ciphertext = '28 ZsTpaumnrhsrkotDscf nameggi cit rhi reenfeie  tüd enomntpeikadrtnenbu !g'
>>> key = 'e = 2.718281828459045235360287471352662497757247093699959574966967627724076'

>>> digit_count(key)
70

>>> digit_position(3, 2, key)
24
>>> digit_position(6, 3, key)
37
>>> digit_position(9, 42, key)
-1
```

```
>>> decrypt(ciphertext, key)
'Das komputermaschine ist nicht für der gefingerpoken und mittengraben!'

>>> ciphertext = ' ftLsrnUisyTelvWya les   Kahen  Eu9ocr RRll 5ochyV.!s ynle5ITiRta'
>>> key = '6630295(160n897o0338533q1=884868J7Z624$fJ035W3375E13016898B4X9.74'
>>> decrypt(ciphertext, key)
'Listen very carefully. I shall say this only once!'
"""

def digit_count(text):

    return sum(character.isdigit() for character in text)

def digit_position(digit, occurrence, text):

    digit = str(digit)
    position = -1
    for _ in range(occurrence):
        position = text.find(digit, position + 1)
        if position == -1:
            return -1

    return position

def decrypt(ciphertext, key):

    assert len(key) == len(ciphertext), 'invalid key'

    count = digit_count(key)
    plaintext = ''
    index = 0
    while len(plaintext) < count:
        occurrence, digit = index // 10 + 1, index % 10
        position = digit_position(digit, occurrence, key)
        if position != -1:
            plaintext += ciphertext[position]
        index += 1

    return plaintext

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[:-1], start=1)) % 11

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

    return (
        isinstance(code, str)                # code is a string
        and len(code) == 10                  # code contains 10 characters
        and code[:-1].isdigit()              # first nine characters of the code are digits
        and check_digit(code) == code[-1]    # check digit is valid
    )

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
            [:-1]))

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

    return (
        isinstance(code, str)                # code is a string
        and len(code) == 13                  # code contains 13 characters
        and code.isdigit()                   # code only contains digits
        and check_digit(code) == code[-1]    # check digit is valid
    )

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]
    """

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

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)

    return (
        isinstance(code, str)                # code is a string
        and len(code) == 13                  # code contains 13 characters
        and code[:3] in {'978', '979'}       # code has valid prefix
        and code.isdigit()                   # code only contains digits
        and check_digit(code) == code[-1]    # check digit is valid
    )

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 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)

    return (
        isinstance(code, str)                # code is a string
        and len(code) == 13                  # code contains 13 characters
        and code[:3] in {'978', '979'}       # code has valid prefix
        and code.isdigit()                   # code only contains digits
        and check_digit(code) == code[-1]    # check digit is valid
    )

def remove_tags(s):

    """
    Removes all XML tags from the given string and then removes all leading and trailing
        whitespace.
    NOTE: there are also specified modules for it like BeautifulSoup

    >>> 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 Python expression to creates a new object with the same
            internal state as self
        return f'ISBN13({int(self.code)}, {self.length})'

    def isvalid(self):

        def check_digit(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 a string
            return str((10 - check) % 10)

        # check validity of check digit
        return self.code[12] == check_digit(self.code)
```

```
    def asISBN10(self):

        def check_digit(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 a string
            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 = check_digit(code)
        return f'{code[:self.length]}-{code[self.length:]}-{check}'

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