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 do not use the name "sum" for the variable "sum" because this is the
#       name of a built-in function in Python
total = term1 + term2

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

1.3 Dogarithmic scale

```
import math

# read a dog's age
dog_age = float(input())

# compute corresponding human age
human_age = 16 * math.log(dog_age) + 31

# print corresponding human age
print(human_age)
```

1.4 The diatomist

```
import math

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

# estimate number of smaller circles that fit in larger circle
```

```
count = math.floor(0.83 * (R ** 2 / r ** 2) - 1.9)

# determine coverage of larger circle
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')
```

1.5 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')
```

1.6 Clock hands

```
# read time on 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 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 correctness of 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 concerning 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 (x,y)-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 both coin throws; outcomes are converted to Boolean values
# (head -> True, tail -> False) in order to ease the implementation
coin1 = input() == 'head'
coin2 = input() == 'head'

# 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('head' if coin1 else 'tail')

# output response of second scientist
print('head' if coin2 else 'tail')
```

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 or not knight can jump 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 Russian peasants

```
# read multiplication factors
factor1 = int(input())
factor2 = int(input())

# generate Russion multiplication table while keeping track of the sum that will
# finally give the product
product = 0
while factor2 > 0:
    print(factor1, factor2)
    if factor2 % 2:
        product += factor1
    factor1, factor2 = factor1 * 2, factor2 // 2

# output the product
print(product)
```

3.3 Payslip

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

# initialize total salary with random number
total = random

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

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

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

    # read salary of next worker
    salary = input()

# output average salary
print(f'average salary: {(total - random) / workers:.2f}')
```

3.4 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')
```

3.5 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)
```

3.6 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 in sequence 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
    check_digit = sum((i + 1) * int(code[i]) for i in range(9)) % 11
    """

    # extract check digit from ISBN-10 code
    x10 = code[-1]

    # check whether computed and extracted check digits are the same
    if (check_digit == 10 and x10 == 'X') or x10 == str(check_digit):
        print('OK')
    else:
        print('WRONG')

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

4.2 Scrambled images

```
# read separator
separator = input()

# read number of lines in the given image
lines = int(input())

# process encoded lines one by one
for _ in range(lines):

    # read encoded line
    line = input()

    # decode line
    index = line.index(separator)
    line = line[index + 1:] + line[:index]

    # output decoded line
    print(line)
```

4.3 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)
"""
```

4.4 Word evolutions

```
# 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
from string import ascii_uppercase
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 double the 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
    slice = alphabet[pos:pos + width]

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

4.5 Mathemagical

```
# read number
number = input()

# determine length of number
length = len(number)

# determine sum by suppressing each of the digits in the given number
total = 0
for index in range(length):
```

```
    term = int(number[:index] + number[index + 1:])
    total += term
    print(f'{"+" if index == length - 1 else " "}{term:{length}d}')

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

4.6 Alchemical reduction

```
# read polymer
polymer = input()

# reduce polymer
reduction = ""
for unit in polymer:
    if reduction and reduction[-1] == unit.swapcase():
        reduction = reduction[:-1]
    else:
        reduction += unit

# output completely reduced polymer
print(f'{reduction}({len(reduction)})')
```

5 Series 05: functions

5.1 ISBN

```
def isISBN(code):

    """
    Return True if the argument is a string that contains 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) and      # code must be a string
        len(code) == 10 and            # code must contain 10 characters
        code[:9].isdigit() and         # first nine characters must be digits
        checkdigit(code) == code[-1]   # 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('997150210')
    '0'
    >>> checkdigit('938389293')
    '5'
    """

    # compute check digit
    check = sum((i + 1) * int(code[i]) for i in range(9)) % 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('lion', 'tiger')
    ('jeopard', 'laguar')
    >>> hybridize('schnauzer', 'poodle')
    ('schnoodle', 'pauzer')
    """

    # 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 Jupiter-C

```
def reduce(secret):

    """
    >>> reduce('HUNTSVILLEX')
    'HUNTSVILEX'
    >>> reduce('TRICHINOPHOBIA')
    'TRICHNOPBA'
    """

    # only keep first occurrence of each letter (case insensitive)
    reduced = ''
    for character in secret:
        if character.upper() not in reduced.upper():
            reduced += character

    return reduced

def key(secret):

    """
    >>> key('HUNTSVILLEX')
    'XHUNTSVILE'
    >>> key('TRICHINOPHOBIA')
    'ATRICHNOPB'
    """

    # deduplicate secret
```

```
    secret = reduce(secret)
    assert len(secret) == 10, 'invalid key'

    # rotate secret to the right (puts letter that corresponds to zero upfront)
    return secret[-1] + secret[:-1]

def encode(serial_number, secret):

    """
    >>> encode(29, 'HUNTSVILLEX')
    'UE'
    >>> encode(63, 'TRICHINOPHOBIA')
    'NI'
    """

    # convert secret to key
    secret = key(secret)

    # replace all digits in serial number by their corresponding letters
    encoded = ''
    for digit in str(serial_number):
        encoded += secret[int(digit)]

    # return encoded serial number
    return encoded

def decode(encoded, secret):

    """
    >>> decode('UE', 'HUNTSVILLEX')
    29
    >>> decode('NI', 'TRICHINOPHOBIA')
    63
    """

    # convert secret to key
    secret = key(secret)

    serial_number = ''
    for letter in encoded:
        serial_number += str(secret.index(letter))

    # return original serial number
    return int(serial_number)

def next(encoded, secret):

    """
    >>> next('UE', 'HUNTSVILLEX')
    'NX'
    >>> next('NI', 'TRICHINOPHOBIA')
    'NC'
    """

    return encode(decode(encoded, secret) + 1, secret)

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

5.4 Persistence

```
def multiplication(number):

    """
    >>> multiplication(327)
    42
```

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

    # multiply all digits of the number together
    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
    """
```

```
    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()
```

5.5 Mlecchita vikalpa

```
import string

def iskey(key1, key2):

    """
    >>> iskey('THEQUICKBROWN', 'FXJMPSVLAZYDG')
    True
    >>> iskey('ABCDEFGHIJKLM', 'NOPQRSTUVW???')
    False
    >>> iskey('ABCDEFGHIJKLM', 'NOPQRSTUVW')
    False
    """

    # both arguments must be strings of length 13
    if len(key1) != 13 or len(key2) != 13:
        return False

    # each character of the alphabet must occur just once in the keys
    letters = (key1 + key2).upper()
    for letter in string.ascii_uppercase:
        if letter not in letters:
            return False

    return True

def encode_character(character, key1, key2):

    """
    >>> encode_character('Q', 'THEQUICKBROWN', 'FXJMPSVLAZYDG')
    'M'
    >>> encode_character('v', 'THEQUICKBROWN', 'FXJMPSVLAZYDG')
    'c'
    >>> encode_character('?', 'THEQUICKBROWN', 'FXJMPSVLAZYDG')
    '?'
    """

    # convert keys to uppercase
    key1, key2 = key1.upper(), key2.upper()

    # swap keys such that character never occurs in second key
    if character.upper() in key2:
        key1, key2 = key2, key1

    # characters that are no letters remain unchanged
    if character.upper() not in key1:
        return character

    # encode letter (uppercase)
    encoded_letter = key2[key1.index(character.upper())]
```

```
    # maintain uppercase and lowercase
    return encoded_letter if character.isupper() else encoded_letter.lower()

def encode(message, key1, key2):

    """
    >>> encode('A person who does nothing will enjoy no happiness.', 'THEQUICKBROWN', '
        FXJMPSVLAZYDG')
    'B ujziyg dxy wyji gyfxsgn dskk jgeyo gy xbuusgjii.'
    """

    return ''.join(
        encode_character(character, key1, key2) for character in message
    )


# NOTE: encoding and decoding are the same
decode = encode

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

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

    # remove dashes from the given 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 given code
    return checkdigit(code) == code[-1]

def checkdigit(code):

    """
    >>> checkdigit('997150210')
    '0'
    >>> checkdigit('938389293')
    '5'
    """

    # compute check digit
    check = sum((i + 1) * int(code[i]) for i in range(9)) % 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 given 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((i + 1) * int(code[i]) for i in range(9)) % 11

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

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

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

    # check whether first nine characters of given 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 given 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 i % 2 else 1) * int(code[i]) for i in range(12))

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

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

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

```
    # check whether first nine characters of given 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:

        if isinstance(code, str):

            if isbn13 is None:
                checks.append(isISBN(code, len(code) == 13))
            else:
                checks.append(isISBN(code, isbn13))

        else:

            checks.append(False)

    # 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 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 the check digit
        check = sum((3 if i % 2 else 1) * int(code[i]) for i in range(12))

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

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

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

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

    # check whether all characters of the given 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 the check digit
        check = sum((3 if i % 2 else 1) * int(code[i]) for i in range(12))

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

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

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

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

    # check whether first nine characters of the given 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 given 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 displayBookInfo(code):

    """
    >>> displayBookInfo('9780136110675')
    Title: The Practice of Computing using Python
    Authors: William F Punch, Richard Enbody
    Publisher: Addison Wesley
    >>> displayBookInfo('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:

        # open web page with URL of imitated ISBNdb.com that provides
        # information about the book with the given ISBN-13 code
        url = f'http://pythia.ugent.be/cgi-bin/isbn9/books.cgi?value1={code}'
        info = urllib.request.urlopen(url)

        # extract and output selected book information from XML
        for line in info:
            line = line.decode('utf-8')
            if line.startswith('<Title>'):
                print('Title: {}'.format(remove_tags(line)))
            elif line.startswith('<AuthorsText>'):
                print('Authors: {}'.format(remove_tags(line).rstrip(', ')))
            elif line.startswith('<PublisherText '):
                print('Publisher: {}'.format(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) and 1 <= length <= 5, 'invalid ISBN code'

        # object properties: ISBN-code and length of country group
        # convert to string of 13 characters with leading zeros
        self.code = str(code).zfill(13)
        self.length = length

    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 results in 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 i % 2 else 1) * int(code[i]) for i in range(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((i + 1) * int(code[i]) for i in range(9)) % 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 str(self.code)[:3] != '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()
```