Division of Computing Science
Department of Information Technology
Uppsala University

Re-Exam in Programming in Python (1TD327), 5 ECTS

- *Note*: Answers should be written in English. All problems must be solved using Python code. Use short, appropriate and descriptive names for all your variables and functions. Note that your grade will be affected negatively if:
  - your code is unclear and/or hard to read
  - identical snippets of code are repeated several times (copy/paste)
  - common data structures (lists, dictionaries, strings) are not used correctly
- *Tools*: Any electronic devices or any other formula books **are not allowed**!
- *Date*: August 17, 2022, 14:00 – 19:00.
- *Place*: Bergsbrunnagatan 15, sal 2
- *The grading system:*
  - The exam has two parts: (A1.-A10.) Basic and (B1.-B16.) Advanced.
  - In order to **Pass**, you need to get at least **75%** of the points from Part (A).
  - If Part (A) is failed, part (B) will not be graded.
  - To get **Grade 4**, you need to pass part (A) and get at least **50%** of the points from part (B).
  - To get **Grade 5**, you need to get at least **85%** of the points **both** from part (A) and (B).

Part A

Estimated time required: 1 hour

Each correct answer is worth 1 point.

1. A module written in Python is ...
   Select one alternative:
   A. A file with the extension `.pym` that can contain functions, classes and can import other modules, but no scripting.
   B. A file with the extension `.py` that can contain functions, classes and can import other modules, but no scripting.
   C. A synonym for "package" which can be installed with the package manager `pip` or `conda`.
   D. Any file with either the extension `.py` or the extension `.pym` containing any kind of Python code.

2. What is the output of the code snippet below? Code:

```
bingo = ['99', '17', '8', '54', '-99',  '1']

n = 0
while n < len(bingo):
  if n == 0:
    result = bingo[n]
  else:
    result += bingo[n]
  n += 1

print(result, type(result))
```

   Select one alternative:
   A. `80 <class 'int'>`
   B. `80 <class 'str'>`
   C. `9917854-991 <class 'str'>`
   D. `TypeError`

3. What is printed on the screen if the code below is executed and why?
   Code:

```
cake = {'eggs': 4, 'sugar': 200, 'flour': 350, 'butter': 250, 'lemon':
   3, 'salt': 0}

pinch_of_salt = 0.2

print(f'Stock of ingredients for the cake: {cake}')
print(f'Quantity of salt to add: {pinch_of_salt}')
```

```
def add_pinch_of_salt():
    pinch_of_salt = 0.5
    cake['salt'] += pinch_of_salt

add_pinch_of_salt()

print(f'Updated list of ingredients for the cake: {cake}')
print(f'Quantity of salt added: {pinch_of_salt}')
```

Select one alternative:

A. It is forbidden in Python to overwrite an integer by a float:

```
TypeError
```

B. Dictionaries are mutable objects in Python and floats are immutable such that `cake` is modified in place by the function `add_pinch_of_salt()` as follows, but `pinch_of_salt` is unchanged:

```
Stock of ingredients for the cake: {'eggs': 4, 'sugar': 200, '
   flour': 350, 'butter': 250, 'lemon': 3, 'salt': 0}
Quantity of salt to add: 0.2
Updated list of ingredients for the cake: {'eggs': 4, 'sugar':
   200, 'flour': 350, 'butter': 250, 'lemon': 3, 'salt': 0.5}
Quantity of salt added: 0.2
```

C. Dictionaries are immutable objects in Python such that `cake` cannot be updated:

```
Stock of ingredients for the cake: {'eggs': 4, 'sugar': 200, '
   flour': 350, 'butter': 250, 'lemon': 3, 'salt': 0}
Quantity of salt to add: 0.2
Updated list of ingredients for the cake: {'eggs': 4, 'sugar':
   200, 'flour': 350, 'butter': 250, 'lemon': 3, 'salt': 0}
Quantity of salt added: 0.5
```

D. Dictionaries are mutable objects in Python and floats are mutable too such that `cake` and `pinch_of_salt` are modified by the function `add_pinch_of_salt()` as follows:

```
Stock of ingredients for the cake: {'eggs': 4, 'sugar': 200, '
   flour': 350, 'butter': 250, 'lemon': 3, 'salt': 0}
Quantity of salt to add: 0.2
Updated list of ingredients for the cake: {'eggs': 4, 'sugar':
   200, 'flour': 350, 'butter': 250, 'lemon': 3, 'salt': 0.5}
Quantity of salt added: 0.5
```

4. Which of the following statements is true for the two variables named `str1` and `str2` in the following code?
   Code:

```
str1 = 'Functions are important programming concepts.'

def print_fn():
    str2 = 'Variable scope is an important concept.'
    print(str2)
```

Select one alternative:

A. Variables are assumed to be global unless they are explicitly declared to be local in a function, therefore both `str1` and `str2` are global variables.

B. All variables defined within a function are global. You cannot define a local variable in a function. Therefore, `str1` is a local variable and `str2` is a global one.

C. Variables are assumed to be local unless they are explicitly declared to be global in a function. Therefore `str1` is a global variable and `str2` is local to the function `print_fn()`.

D. All variables defined within a script are local, therefore `str1` and `str2` are local variables.

5. What is the output of the following code?
Code:

```
pairs = [[n,s] for n in range(4) for s in 'abcd' if n % 2 == 0]

print(pairs)
```

Select one alternative:

A. `[[0, 'a'], [0, 'b'], [0, 'c'], [0, 'd']]`

B. `[[2, 'a'], [2, 'b'], [2, 'c'], [2, 'd']]`

C. `IndexError: out of range`

D. `[[0, 'a'], [0, 'b'], [0, 'c'], [0, 'd'], [2, 'a'], [2, 'b'], [2, 'c'], [2, 'd']]`

6. Given the following definition of the class `Dice`:

```
import random

class Dice:
    def __init__(self, sides):
        self.sides = sides
        self.value = random.randint(1, self.sides)

    def __str__(self):
        return f'Sides: {self.sides:2d}, value: {self.value:2d}'

    def roll(self):
        self.value = random.randint(1, self.sides)
```

Which statement correctly classifies the class components? Select one alternative:

A. methods: `self.value`, `self.sides`
attributes: `sides`, `self`
constructor: `roll`

B. methods: `self.value`, `self.sides`
attributes: `__init__`, `__str__`
constructor: `roll`

C. methods: `__init__`, `__str__`, `roll`
attributes: `sides`, `value`
constructor: `__init__`

D. methods: `self`, `sides`
attributes: `__init__`, `__str__`
constructor: `Dice()`

7. Which of the following code snippets correctly uses the class `Dice` as defined in previous question? Select one alternative:

A.
```
d1=Dice(6)
print(d1.sides(), d1.values())
```

B.
```
d1=Dice(6)
d2=Dice(10)

if(d1.sides>d2.sides):
    d1.roll()
else:
    d2.roll()
```

C.
```
d1=Dice()

for k in range(0, 10):
    d1.roll()
    print(d1)
```

D.
```
d_list=[Dice(10) for i in range(0, 10)]

for d in d_list:
    d.roll()=10
```

8. We keep the inventory of a candy shop in a dictionary as a collection of candy names and amounts in grams.

```
inventory={"Bilar" : 2000, "Djungelvral": 5000, "Nappar": 1000, "
   Gelehallon": 2000, "Colaflaskor": 4000 }
```

Which function listed below can be used to reduce the amount of a candy type in inventory after a purchase? The following code snippet illustrates a call to such a function.

```
reduce_inventory(inventory, 'Bilar', 300)
```

Select one alternative:

A.
```
def reduce_inventory(inventory, candy, amount):
    for item in inventory:
        if(item==candy):
            item=item - amount
```

B.
```
def reduce_inventory(inventory, candy, amount):
    inventory.candy=inventory.candy - amount
```

C.
```
def reduce_inventory(inventory, candy, amount):
    items=inventory.items()
    for pair in items:
        if(pair[0]==candy):
            pair[1]=pair[1] - amount

    inventory=dictionary(items)
```

D.
```
def reduce_inventory(inventory, candy, amount):
    inventory[candy]=inventory[candy] - amount
```

9. Given the following list:

```
client=['Bjorn', 'Svensson', 'Visbygatan', 50, 19534, 'Lund', ['#1
   asdfqw4', '#16sg3s221', '#41as4s6t']]
```

Which statement is invalid and causes an error?
Select one alternative:

A.
```
print(client[0][0])
```

B.
```
print(client(1))
```

C.
```
print(client[3]//10)
```

D.
```
print(client[-1][-1])
```

10. Under what condition will the following program stop running?

```
import random

while(True):
    k=random.randint(0, 100)
    if(k>20):
        break
    else:
        continue
```

Select one alternative:

A. The program will never stop running.

B. The program will stop running after 100 iterations of the while loop.

C. The program will stop running if the number stored in `k` is larger than 20.

D. The program will stop running if the number stored in `k` is less than or equal to 20.

Part B

Baking pizza

ESTIMATED TIME REQUIRED: 2 HOURS

Write a program that bakes a pizza with different ingredients, according to the instructions given below. You are expected to complete the skeleton of a Pizza module given below:

```
import time

# Class definition.
class Pizza:
    def __init__(...):
    ...
    def startBaking(...):
    ...
    def isReady(...):
```

To help you illustrate how the class could be used here is an example of class usage in a script:

```
myDeliciousPizza = Pizza('wholegrain', 'cheddar', 'sicilian')
# Check the status of my pizza.
time.sleep(3) # wait for 3 seconds after the class was instantiated
myDeliciousPizza.isReady()
```

Tasks to solve

Note that the task B.6. is a bonus task, that means the points you can get with it are not taken into account for defining the grading thresholds.

B.1. [2 points]

Construct a class `Pizza` which accepts 3 string arguments into the constructor: `flour`, `topping` and `crust`, and saves them as class attributes.

B.2. [4 points]

Inside the constructor of `Pizza`, check whether:

- the provided flour is either `'white'`, `'brown'` or `'wholegrain'`,
- the provided topping is either `'mozzarella'`, `'cheddar'` or `'gorgonzola'`,
- the provided crust is either `'thin'`, `'sicilian'` or `'focaccia'`.

Only if all three conditions are true, start baking the pizza by executing a method `startBaking()`. Otherwise abort the pizza making by printing `"Pizza baking aborted, one of the input ingredients is not in stock."`

B.3. [2 points]

Implement a class method `startBaking()`. The method sets an attribute `ready` to 0 and sets an attribute `time_start` to the current time. Current time can be stored into a variable by invoking function `time()` from standard module `time`.

B.4. [3 points]

The purpose of storing the time through `startBaking()` is to check whether 5 minutes have already passed from the point of executing `startBaking()`. Implement this way of checking whether the pizza is already baked, in another class method `isReady()`.

- If the pizza is already baked, the function `isReady()` should set an attribute `ready` to 1, and give an output in the sense:

```
"Pizza with cheddar topping, wholegrain flour dough and sicilian
   crust is ready!"
```

  where the printed topping, flour and crust depends on the choice made at the construction of the class.

- If the pizza is not baked yet, the function `isReady()` should give an output in the sense:

```
"Pizza with cheddar topping, wholegrain flour dough and sicilian
   crust is  still in the oven!
Wait for another 4.180318586031596 minutes".
```

  The time left until the pizza is baked can be computed through the difference `time_start - time.time()`.

B.5. [5 points]

Write a class `Pizzeria`, which takes in an integer `N` (for example N=5), and makes N pizzas (by using the class `Pizza` in its constructor). You can pick the pizza ingredients in any way you want. The instantiated pizzas should be stored into an attribute of the type list with a name `allPizzas`.

Add a class method that will calculate the price of pizzas being prepared. The cost of 1 pizza is the sum of the ingredient prices used to make the pizza. Given is the list of prices below:

- flours: white costs 10 units, brown costs 15 units, wholegrain costs 20 units,
- toppings: mozzarella costs 3 units, cheddar costs 6 units, gorgonzola costs 9 units,
- crusts: thin costs 2 units, sicilian costs 4 units, focaccia costs 7 units

The function should print out the total price of `N` pizzas in the sense: `"The total costs for 5 ordered pizzas is XXX units."`

If one of the ingredients for a given pizza is not available (you can assume that the list of prices is the list of available ingredients) reject the order by printing out: `"One of the desired ingredients is not available. Order aborted."`

B.6. [3 points] Bonus task

Extend the `Pizza` class by adding a functionality that allows extra toppings to be added after the baking of pizza has already started. The only allowed extra toppings can be 'prosciutto', 'funghi' or 'pineapple'. Allow the addition of an extra topping only if the pizza has not been fully baked yet.

Equation parsing

Estimated time required: 45 minutes

Task to solve

B.7. [4 points]

Write a function that for a given string checks if it is a valid equation. An equation is valid if it satisfies the following conditions:

- An equation can contain only numbers and characters "+" and "=".
- Equal sign can occur only once.
- To the left and to the right of the equal sign must be a number.
- To the left and to the right of every "+" sign must be a number.

Examples of strings that are equations:

- `"1+2=456"`
- `"12+34=40+3+2+01"`
- `"123=143"`
- `"12+3=1+23"`
- `"123=103+20"`
- `"0+00+000+00100=00000+000000"`

Examples of strings that are not equations:

- `"1+2=3=2+1"`
- `"12=3*4"`
- `"a+1=1+a"`
- `"12++3=15"`
- `"tralala"`
- `"1 + 2 = 3"`
- `"c#"`
- `"1+2+=3"`
- `"="`

A river of spaces

Estimated time required: 1 hour 15 minutes

We are given a block of text printed in such a way that every character, including spaces and punctuation marks, has the same width. The text is printed out over several lines. Sometimes it happens that the space characters in consecutive lines align in a way that forms a river of spaces. If we look at such a block of text from a distance the river appears as a notable white patch stretching across the text.

Task to solve

B.8. [5 points]

We will define the river of spaces to be a sequence of space characters in consecutive lines of text where a space character in a line below lies either in the same position or one position to the left or right of the space character in the line above. Whitespace to the right of the rightmost character in a line cannot be a part of a river of spaces.

Write a function that gets as an input a text block given as a list of lines and outputs the length of the longest river of spaces contained in the text block.

The longest river of spaces in the following text runs from line five to line fourteen and has length 10. The longest river of spaces is emphasized by replacing the space characters with the sequence `| |`.

```
Two fierce and enormous bears, distinguished
by the appellations of Innocence and Mica
Aurea, could alone deserve to share the
favour of Maximin. The cages of those trusty
guards were| |always placed near the
bed-chamber| |of Valentinian, who frequently
amused his| |eyes with the grateful spectacle
of seeing| |them tear and devour the bleeding
limbs of| |the malefactors who were abandoned
to their| |rage. Their diet and exercises were
carefully| |inspected by the Roman emperor;
and, when| |Innocence had earned her discharge
by a long| |course of meritorious service, the
faithful| |animal was again restored to
the freedom of her native woods.
```

Appendix

Excerpt from documentation for module `random`:

```
choice(self, seq) method of Random instance
    Choose a random element from a non-empty sequence.

gauss(self, mu, sigma) method of Random instance
    Gaussian distribution.

    mu is the mean, and sigma is the standard deviation.  This is
    slightly faster than the normalvariate() function.

    Not thread-safe without a lock around calls.

randint(self, a, b) method of Random instance
    Return random integer in range [a, b], including both end points.

random(...)
    random() -> x in the interval [0, 1).

randrange(self, start, stop=None, step=1, _int=<type 'int'>, _maxwidth
   =9007199254740992L) method of Random instance
    Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

sample(self, population, k) method of Random instance
    Chooses k unique random elements from a population sequence.

    Returns a new list containing elements from the population while
    leaving the original population unchanged.  The resulting list is
    in selection order so that all sub-slices will also be valid random
    samples.  This allows raffle winners (the sample) to be partitioned
    into grand prize and second place winners (the subslices).

    Members of the population need not be hashable or unique.  If the
    population contains repeats, then each occurrence is a possible
    selection in the sample.

    To choose a sample in a range of integers, use xrange as an argument.
    This is especially fast and space efficient for sampling from a
    large population:   sample(xrange(10000000), 60).

shuffle(x)
    Shuffle the sequence x in place.
```

Excerpt from the Python documentation for some the methods on string objects:

```
  center(self, width, fillchar=' ', /)
      Return a centered string of length width.
```

```
    Padding is done using the specified fill character (default is a
 space).

count(...)
    S.count(sub[, start[, end]]) -> int

    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are
    interpreted as in slice notation.
isalpha(self, /)
    Return True if the string is an alphabetic string, False otherwise.

    A string is alphabetic if all characters in the string are
 alphabetic and there
    is at least one character in the string.

isascii(self, /)
    Return True if all characters in the string are ASCII, False
 otherwise.

    ASCII characters have code points in the range U+0000-U+007F.
    Empty string is ASCII too.

isdecimal(self, /)
    Return True if the string is a decimal string, False otherwise.

    A string is a decimal string if all characters in the string are
 decimal and
    there is at least one character in the string.

isdigit(self, /)
    Return True if the string is a digit string, False otherwise.

    A string is a digit string if all characters in the string are
 digits and there
    is at least one character in the string.

isdecimal(self, /)
    Return True if the string is a decimal string, False otherwise.

    A string is a decimal string if all characters in the string are
 decimal and
    there is at least one character in the string.

isdigit(self, /)
    Return True if the string is a digit string, False otherwise.

    A string is a digit string if all characters in the string are
 digits and there
    is at least one character in the string.

islower(self, /)
```

```
    Return True if the string is a lowercase string, False otherwise.

    A string is lowercase if all cased characters in the string are
 lowercase and
    there is at least one cased character in the string.

isnumeric(self, /)
    Return True if the string is a numeric string, False otherwise.

    A string is numeric if all characters in the string are numeric and
 there is at
    least one character in the string.

isupper(self, /)
    Return True if the string is an uppercase string, False otherwise.

    A string is uppercase if all cased characters in the string are
 uppercase and
    there is at least one cased character in the string.

join(self, iterable, /)
    Concatenate any number of strings.

    The string whose method is called is inserted in between each given
 string.
    The result is returned as a new string.

    Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

split(self, /, sep=None, maxsplit=-1)
    Return a list of the substrings in the string, using sep as the
 separator string.

      sep
        The separator used to split the string.

        When set to None (the default value), will split on any
 whitespace
        character (including \\n \\r \\t \\f and spaces) and will
 discard
        empty strings from the result.
      maxsplit
        Maximum number of splits (starting from the left).
        -1 (the default value) means no limit.

    Note, str.split() is mainly useful for data that has been
 intentionally
    delimited.  With natural text that includes punctuation, consider
 using
    the regular expression module.

strip(self, chars=None, /)
```

```
    Return a copy of the string with leading and trailing whitespace
 removed.

    If chars is given and not None, remove characters in chars instead.

upper(self, /)
    Return a copy of the string converted to uppercase.
```