Division of Computing Science
Department of Information Technology
Uppsala University

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*: January 7, 2022, 8:00 – 13:00.
- *Place*: Polackbackens skrivsal.
- *The grading system:*
  - The exam has two parts: (A1.-A10.) Basic and (B1.-B15.) 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. After the following code executes, how does the programmer compute a square root of 2?
   Code:

```
import numpy
```

   Select one alternative:

   A. `np.sqrt(2)`

   B. `sqrt(2)`

   C. `numpy.sqrt(2)`

   D. `sqrt(2).numpy`

2. What is the data type of the variable 'months'?
   Code:

```
months = [i for i in range(1,13)]
```

   Select one alternative:

   A. list

   B. double

   C. string

   D. integer

3. See the code below. After the code is executed, the program prints [4, 2, 3] and 1. Why does the value of countList get overwritten by a function and the value of countInteger does not?
   Code:

```
countList = [1, 2, 3]
def someFunction1():
    countList[0] = 4

someFunction1()
print(countList)

def someFunction2():
    countInteger = 0

countInteger = 1
someFunction2()
print(countInteger)
```

Select one alternative:

A. This is because `countInteger` is an integer which is a mutable type, but `countList` is a list which is an immutable type.

B. This happens because Python is an object-oriented programming language.

C. This happens because `someFunction2` is defined before `countInteger` is set to 1, while `someFunction1` is defined after `countList` is set to `[1, 2, 3]`.

D. This is because `countInteger` is an integer which is an immutable type, but `countList` is a list which is a mutable type.

4. Which parts of the program below are referred to as the global scope, and which is referred to as the local scope?
Code:

```
# Part 1 start #
someList = [2*i+1 for i in range(1,6)]
someList[4]=0
#Part 1 stop #
def someFunction(inp):
    # Part 2 start #
    cvar = 3
    inp[0] = 22*cvar
    return cvar + inp[3]
    # Part 2 stop #
# Part 3 start #
out = someFunction(someList)
print(out)
# Part 3 stop #
```

Select one alternative:

A. Part 1 and Part 3 together are the global scope, Part 2 is the local scope.

B. Part 2 and Part 3 together are the global scope, Part 1 is the local scope.

C. Part 1 and Part 2 and Part 3 together are the local scope, there is no global scope.

D. Part 1 and Part 2 together are the global scope, Part 3 is the local scope.

5. Which of the codes below does correctly compute an approximation to $\pi^2/6$ with the Euler method, by using the list comprehension?
Select one alternative:

A.
```
res = 1
for k in range(1,N+1):
    res = res + 1/k**2
```

B.
```
res = sum([1/k**2 for k in range(1,N+1)])
```

C.

```
listt = [0] * (N+1)
for k in range(1, N+1):
    listt[k] = 1/k**2
res = sum(listt)
```

6. What is the main purpose of defining classes when programming? Select one alternative:

   A. Classes compress the input data.

   B. Classes give a better input/output experience when reading images.

   C. Classes speed up the code.

   D. Classes provide a generalization of parts of code and contribute to the code clarity and reusability.

7. Consider a class defined below. How do we retrieve the output of the method `__str__`? Code:

```
class Breakfast:
    def __init__(self, crunch, liquid):
        self.crunch = crunch
        self.liquid = liquid

    def __str__(self):
        return 'crunch=' + self.crunch + ', liquid=' + self.liquid
```

   Select one alternative:

   A. `print(Breakfast('Cornflakes', 'Milk').str)`

   B. `print(Breakfast('Cornflakes', 'Milk').__str__)`

   C. `print(Breakfast('Cornflakes', 'Milk'))`

   D. `print(Breakfast)`

8. When do we use a dictionary as the data structure in Python?
   Select one alternative:

   A. When the data values have a prescribed order.

   B. When we want to allow the data values to have duplicate entries.

   C. When we wish to associate the data values with keys and, where each key allows for an association with several data values.

   D. When we wish to associate the data values with keys, where each key allows for an association with one data value.

9. Given a predefined list `myList`, what does `myList[-1]` return? Select one alternative:

   A. It returns all the elements in the list, except the second one.

   B. It returns all the elements in the list, except the last one.

C. It returns an error, the syntax is invalid.

D. It returns the last element of that list.

10. Which is the output of the code snippet below?

```
aList = ["Uppsala", [2, 4, 8, 16]]
print(aList[0][3:])
print(aList[1][3])
```

A. `'U', 'sala'` and `2, 16`

B. `'sala'` and `16`

C. `'U', 'sala'` and `4, 16`

D. `IndexError`

Part B

Qwixx Dice Game Simulation

Estimated time required: 2 hours 30 minutes

For this part, you are expected to model a simplified version of the "Qwixx" dice game. The tasks for you to complete are to implement Python classes for representing various kinds of objects that are needed in a Qwixx game, and a script for simulating playing a turn. Make sure to remove the `pass` and the `...` instructions in the code when you implement the different tasks.

Description of the rules for playing Qwixx

Qwixx is usually played by 2-5 players who roll in turn 6 dice for scoring points. Each player has a score sheet with:

- The numbers from 2 to 12 in rows of color red (r) and yellow (y),
- The numbers from 12 to 2 in rows of color green (g) and blue (b).

An illustration of a real Qwixx score sheet is shown on Figure 1.

In order to win the game, each player wants to mark off as many numbers as possible, but you can mark off a number only if it is to the right of all off-marked numbers in the same row. For example on Figure 1:

- In the red row, all numbers ranging from 6 to 12 that can be marked off since they are on the right to the number 5 which is marked off. The numbers 2, 3, 4 cannot be marked off any longer.

- In the blue row, all numbers ranging from 6 to 2 that can be marked off since they are on the right to the number 7 which is the rightmost number that is marked off. The numbers 12, 11, 10, 8 cannot be marked off any longer.

The more marks a player makes in a row, the higher its score for that row.

A turn is played as follows:

1. The active player rolls six dice: 2 white (w_1, w_2) and 1 of each of the 4 colors listed above (r, y, g, b). Then the active player marks off the sum of one colored die and one white die in the row that has the same color as the die. For example, if the dice were rolled as on the Figure 1 ($w_1 = 6$, $w_2 = 5$, $r = 2$, $y = 6$, $g = 1$, $b = 1$), the active player can mark either $w_1 + y = 12$ or $w_2 + y = 11$ in the yellow row, or $w_1 + b = 7$ or $w_2 + b = 6$ in the blue row, etc.

2. Each nonactive player marks off the sum of the 2 white dice on one of their four rows, that is to say $w_1 + w_2 = 11$ either in the red, yellow, green, or blue row.

The score of a player, or number of points, is calculated as the sum of all off-marked numbers. Once all players are done, the turn ends. The player sitting next to the current active player becomes the active player for the next turn. Turns are played until at least one player has a score of 50 points.

Description of the Python classes and objects used for modelling Qwixx

Three classes are used in the game modelling:

- The `Dice` class (code listing 1) that is used for modelling each of the 6 dice used in the game (2 white dice and 4 colored ones). The constructor of the class takes 2 input parameters that are:
  - `sides`: (integer, defaut value = 6), number of faces of the die,
  - `color`: (string, defaut value = `'white'`), color of the die.
- The `ColorRow` class (code listing 3) that models a colored scoring line on a score sheet, constructed from the following input parameters:
  - `color`: (string) color of the row,
  - `sort_desc`: (boolean) indicates whether the numbers in the row should be sorted in descending order or in ascending one.
- The `Player` class (code listing 5) that models a player by its name and its score sheet. The constructor takes 1 input parameter:
  - `name`: (string) name of the player.

Figure 1: Example of a score sheet of a player in a Qwixx game

Tasks to solve

B.1. [1 point]

Implement the constructor of the `Dice` class:

- Define an attribute `sides`: integer which has the value 6.
- Define an attribute `color` from the input parameter *color*.
- Define an attribute `face`: integer that shows one random face.

B.2. [2 points]

Implement the `__str__(self)` method of the `Dice` class, based on the output example in the code listing 2.

B.3. [2 points]

Implement `roll(self)` method of the `Dice` class, based on the output example in the code listing 2. This method assigns a new random value to the `face` attribute.

B.4. [1 point]

Implement the constructor of the `ColorRow` class:

- Define an attribute `color` from the *color* input argument.

- Define an attribute `sort_desc` that is assigned to the *sort_desc* input argument.
- Define an attribute `numbers`: a list of numbers ordered as shown on Figure 1.
- Define an attribute `offmarks`: a list for the numbers marked in each row (nothing is marked off at the beginning of the game).

B.5. [2 points]

Implement `markable(self)` method of the `ColorRow` class, based on the output example in the code listing 4 and the `if` statement already written. This method calculates and returns the list of numbers that can be marked off, that is to say, the numbers that are to the right of the maximum or minimum number crossed in the row.

B.6. [2 points]

Implement the `mark(self, num)` method of the `ColorRow` class, based on the output example in the code listing 4. This method places the marked number `num` in the `offmarks` attribute.

B.7. [2 points]

Implement `points(self)` method of the `ColorRow` class, based on the output example in the code listing 4. This method returns the score for the color row as the sum of the numbers that are marked off in this row.

B.8. [1 point]

Implement the constructor of the `Player` class, based on the output example in the code listing 6.

- Define an attribute `name` from the *name* input parameter.
- Define an attribute `rows` as a dictionary in which the keys are the colors 'green', 'red', 'blue', 'yellow', and the values are instances of the `ColorRow` class with the corresponding color. The numbers in the row must be sorted as on Figure 1 for each color.

B.9. [2 points]

Implement the `rolls(self, dicedict)` method of the `Player` class, based on the output example in the code listing 10. This method should roll all the dice in the dictionary *dicedict*.

B.10. [2 points]

Implement the method `plays(self, val, hue)` of the `Player` class, based on the output example in the code listing 10. This method returns `True` if the player can mark off the number *val* in the row having the color *hue*, else `False` if the player cannot mark off any number.

B.11. [2 points]

Implement the `tot_points(self)` method of the `Player` class, based on the output example in the code listing 10. This method returns the number of points of the player as the sum of the scores of all color rows.

B.12. [3 points]

Create dice and players for simulating a turn by completing the code listing 8: define a set of 6 dice accordingly to the Qwixx description, as well as a list of 3 players with names "Rock", "Paper", and "Scissors".

B.13. Bonus task! [4 points]

Simulate a game with the objects instantiated in the previous task. You can assume that all classes and functions are written and saved together in the same file. Complete the code listing 9: play turns until one of the players has a score of at least 50 points. You are allowed to directly use the function `play_turn(...)` (code listing 7) if you want. Display how many turns were played when the game ends.

Listing 1: Dice class

```
import random

class Dice():
  """ Represents a colored 6-face die """
  def __init__(self, color='white'):
        # B.1.
        pass

  def __str__(self):
        # B.2.
        pass

  def roll(self):
        # B.3.
        pass
```

Listing 2: Dice instance example

```
green_die = Dice(color='green')
print('Die: ', green_die)
print('Rolling the die...')
green_die.roll()
print('Die: ', green_die)

# Output:
Die:  color: green, value: 2
Rolling the die...
Die:  color: green, value: 6
```

Listing 3: ColorRow class

```
class ColorRow():
  """ Represents a colored row with numbers in (2, 12) """
  def __init__(self, color, sort_desc=False):
    # B.4
    pass

  def markable(self):
    # B.5
    if self.sort_desc and self.offmarks != []:  # 12 comes first, only
   smallest values are playable
      most_right = min(self.offmarks) - 1
      num_set = [n for n in range(2, most_right + 1)]

    ...

    return sorted(num_set, reverse=self.sort_desc)

  def mark(self, num):
    # B.6
    pass

  def points(self):
    # B.7
    pass

  def __str__(self):
    line = [str(i) if i not in self.offmarks else 'X' for i in self.
   numbers]
    return f'{self.color} row: \t{line} \t Score = {self.points()}'
```

Listing 4: ColorRow instance example

```
red_row = ColorRow('red', sort_desc=False)
```

```
print(red_row)
print('Offmarks: ', red_row.offmarks)
print('Free numbers: ', red_row.markable())
print()
red_row.mark(5)
print(red_row)
print('Offmarks: ', red_row.offmarks)
print('Free numbers: ', red_row.markable())

# Output:
red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
Offmarks:  []
Free numbers:  [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

red row:  ['2', '3', '4', 'X', '6', '7', '8', '9', '10', '11', '12']
   Score = 5
Offmarks:  [5]
Free numbers:  [6, 7, 8, 9, 10, 11, 12]
```

Listing 5: Player class

```
class Player():
  """ Represents a Qwixx player """
  def __init__(self, name):
    # B.8.
    pass

  def __str__(self):
    return f'''\nPlayer "{self.name}": \r
  {self.tot_points()} points \r
  \rColor rows: \r
  {str(self.rows['green'])}\r
  {str(self.rows['red'])}\r
  {str(self.rows['blue'])}\r
  {str(self.rows['yellow'])}\r
  '''

  def rolls(self, dicedict):
    # B.9
    pass

  def plays(self, val, hue):
    # B.10
    pass

  def tot_points(self):
    # B.11
```

```
    pass
```

Listing 6: Player instance example

```
p = Player('Bob')
print(p)

# Output:
Player "Bob":
  0 points
Color rows:
  green row:  ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
      Score = 0
```

Listing 7: Function for playing a turn

```
def play_turn(dice, idx_active, list_players):
  """
  Simulate playing a turn of Qwixx.
  Random naive strategy (without looking at the colored rows and without
   passing unless no mark possible)
  """
  # Active player starts
  for p in list_players:
    if list_players.index(p) == idx_active:
      print(f'\nActive player "{p.name}" plays')
      # Active player rolls the all dice
      p.rolls(dice)
      print('Dice:')
      for d in dice:
        print(d, ' ', dice[d].face)

      # Active  player picks 1 color + 1 white dice and marks off
      colpick = random.choices(['green', 'red', 'yellow', 'blue'])[0]
      whipick = random.choices(['white1', 'white2'])[0]
      hit = p.plays(dice[whipick].face + dice[colpick].face, colpick)  #
   sum of 2 dice is played on the same row
      print(p)

  # Other players:
```

```
  for p in list_players:
    if list_players.index(p) != idx_active:
      print(f'\nPlayer "{p.name}" plays')
      pick_dice = ['white1', 'white2']
      colpick = random.choices(['green', 'red', 'yellow', 'blue'])[0]
      hit = p.plays(dice['white1'].face + dice['white2'].face, colpick)  #
    sum of 2 dice is played on the same row
      print(p)

  print('\nEnd of turn'.ljust(80, '.'))

  next_player = (idx_active + 1) % len(list_players)

  return next_player
```

Listing 8: Script for creating players and a set of dice

```
# B.12

dice_set = ...

test_players = ...
active_player = 0
```

Listing 9: Game script

```
scores = [p.tot_points() for p in test_players]

# B.13

print(f'\n\nGame over! {k} turns were played.')
```

Listing 10: Playing turn example

```
Turn 1


Active player "Rock" plays
Dice:
white1   3
white2   5
green   4
red   4
yellow   2
blue   1
```

```
Player "Rock":
  7 points
Color rows:
  green row:  ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', '4', '5', '6', 'X', '8', '9', '10', '11', '12']
      Score = 7


Player "Paper" plays

Player "Paper":
  8 points
Color rows:
  green row:  ['12', '11', '10', '9', 'X', '7', '6', '5', '4', '3', '2']
    Score = 8
  red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
      Score = 0


Player "Scissors" plays

Player "Scissors":
  8 points
Color rows:
  green row:  ['12', '11', '10', '9', 'X', '7', '6', '5', '4', '3', '2']
    Score = 8
  red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
      Score = 0


End of turn
   ..............................................................


Turn 2


Active player "Paper" plays
Dice:
```

```
white1   6
white2   4
green   1
red   2
yellow   3
blue   2

Player "Paper":
  13 points
Color rows:
  green row:  ['12', '11', '10', '9', 'X', '7', '6', 'X', '4', '3', '2']
    Score = 13
  red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
      Score = 0


Player "Rock" plays

Player "Rock":
  17 points
Color rows:
  green row:  ['12', '11', 'X', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 10
  red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', '4', '5', '6', 'X', '8', '9', '10', '11', '12']
      Score = 7


Player "Scissors" plays

Player "Scissors":
  8 points
Color rows:
  green row:  ['12', '11', '10', '9', 'X', '7', '6', '5', '4', '3', '2']
    Score = 8
  red row:  ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
   Score = 0
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
      Score = 0


End of turn
   ..............................................................
```

```
...

Turn 13


Active player "Rock" plays
Dice:
white1   2
white2   6
green   2
red   6
yellow   4
blue   3

Player "Rock":
  37 points
Color rows:
  green row:  ['12', '11', 'X', '9', '8', '7', 'X', '5', '4', '3', '2']
    Score = 16
  red row:  ['2', '3', '4', '5', '6', '7', 'X', '9', '10', '11', '12']
   Score = 8
  blue row:   ['12', '11', '10', '9', '8', '7', 'X', '5', '4', '3', '2']
    Score = 6
  yellow row:   ['2', '3', '4', '5', '6', 'X', '8', '9', '10', '11', '12']
      Score = 7


Player "Paper" plays

Player "Paper":
  50 points
Color rows:
  green row:  ['12', '11', '10', '9', 'X', '7', '6', 'X', '4', '3', '2']
    Score = 13
  red row:  ['2', '3', '4', '5', '6', '7', 'X', '9', '10', '11', '12']
   Score = 8
  blue row:   ['X', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 12
  yellow row:   ['2', '3', '4', '5', 'X', '7', '8', '9', '10', 'X', '12']
      Score = 17


Player "Scissors" plays

Player "Scissors":
  43 points
Color rows:
  green row:  ['12', '11', '10', '9', 'X', 'X', '6', '5', '4', '3', '2']
    Score = 15
```

```
  red row:  ['2', '3', '4', 'X', '6', '7', '8', '9', '10', 'X', '12']
   Score = 16
  blue row:   ['12', '11', '10', '9', '8', '7', '6', '5', '4', '3', '2']
    Score = 0
  yellow row:   ['2', '3', 'X', '5', '6', '7', 'X', '9', '10', '11', '12']
      Score = 12


End of turn
   ....................................................................


Game over! 13 turns were played.
```

Medieval volume measurement system

Estimated time required: 45 minutes

In 13th century England the following unit system was in place to measure the volume of liquids:

1 gallon	=	2 pottles
1 pottle	=	2 quarts
1 quart	=	2 pints
1 pint	=	2 chopins
1 chopin	=	2 gills

Task to solve

B.14. [5 points]

Knowing that 1 gallon equals 4.546 liters, write a function `convert_liters_to_UK` that takes the amount given in liters as a floating point value and finds its equivalent in medieval units.

When converting, you need to keep in mind the fact that the amounts in liters specified by floating point numbers have finer granularity than the amounts expressible by the medieval system. As an example we can take 4.547 l, which is equivalent to 1 gallon + 1 mL and therefore cannot be expressed directly with medieval units. The equivalent volume would then be 1 gallon + 1 gill, that is: the smallest volume expressible with medieval units that is large enough to contain the volume given in liters.

The output of your function should be text printed to the terminal as illustrated below.

Example 1

Input:

```
convert_liters_to_UK(4.547)
```

Output:

```
4.547l fits into:
1 gallon
1 gill
```

Example 2

Input:

```
convert_liters_to_UK(1)
```

Output:

```
1l fits into:
1 quart
```

Example 3

Input:

```
convert_liters_to_UK(90)
```

Output:

```
90l fits into:
19 gallons
1 pottle
1 quart
1 chopin
```

Alliterations

Estimated time required: 45 minutes

Alliteration is a repetition of identical initial letter within a group of consecutive words in a sentence. Examples of alliterations are:

- humble house
- rocky road
- potential power play
- Peter Piper

The following examples are not alliterations:

- Water is wet. White is a color.
- cheap and chippy

Task to solve

B.15. [5 points]

Write a function that will determine the number of alliterations in a sentence and determine the number of words in the longest single alliteration. The function should be defined as follows:

```
def analyze_alliterations(sentence):
    # Function body
    return n_alliterations, longest_alliteration
```

The input and output parameters are defined as follows:

- `sentence`: input string containing the sentence to be analyzed. You can assume that punctuation marks have been stripped from the sentence.
- `n_alliterations`: number of alliterations in the sentence
- `longest_alliterations`: the length of the longest alliteration

Example 1

Input:

```
sent1="From a cheap and chippy chopper on a big black block"
print(analyze_alliterations(sent1))
```

Output:

```
(2, 3)
```

Example 2

Input:

```
sent2="To sit in solemn silence in a dull dark dock in a pestilential
   prison with a lifelong lock awaiting the sensation of a short sharp
   shock"
print(analyze_alliterations(sent2))
```

Output:

```
(5, 3)
```

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)

index(self, element) inbuilt list-method in Python
    Searches for a given element from the start of the list and returns
   the lowest index where the element appears.
```