Python Fundamentals
Exercises+Solutions for Python 3.6

1. Write a Python script that prompts the user for a number and displays a message that does indicate if the number is odd or even.
   Note: do consider 0 has being neither odd nor even.
   After having tested a first number, the script should prompt the user for other numbers as long as he or she would like to.

```
while True:
   text=input("Enter an integral number: ")
   nbr=int(text) # Note: this may fail if text is not a valid integral value
   if nbr == 0:
      print ("0 is neither odd nor even")
   elif nbr % 2 == 0: # the remainder of the division of nbr by 2 is 0: nbr is even
      print(nbr, "is an even number")
      # or: print (f"{nbr} is an even number")
      # or: print ("{} is an even number".format(nbr))
      # or: print ("%d is an even number" % (nbr))
   else:
      print (nbr, "is an odd number")
   cont=input("Do you want to test another number (y/n)?")
   if cont not in ("y", "Y"):  # or if cont != "y" and cont != "Y":
      print ("Bye!")
      break # to leave the while loop
```

2. Write a Python script that prompts the user for several numbers (when the user enter the string "stop", the script stop asking for numbers).
   The numbers entered are stored into a list one after the other.
   After having retrieved all the numbers, print the list.
   The script will then computes and print the minimum, the maximum and the mean of the different numbers present in the list.

```
data=[] # or data=list()

while True:
   answer=input("Enter an int or 'stop': ")
   if answer == "stop":
      if len(data) == 0:
         print("At least one value must be provided !")
         continue    # Restart the loop
      else:
         break       # Leave the loop
   data.append(int(answer))

print("Values entered:", data, "Number of values entered:", len(data))
print("Minimum:", min(data), "Maximum:", max(data), "Mean:", sum(data)/len(data))
```

3. Write a Python scripts that implements the “Guess the Number” game.
   The script will generate of a random integral number (use the random module) from 1 to 100, and ask you to guess it.
   The script will tell you if each guess is too high or too low.
   You win if you can guess the number within six tries.

```
import random
secret=random.randint(1,100)

maxNbOfTest=6
currentNbOfTest=1

while currentNbOfTest <= maxNbOfTest:
   #text="Entrer un int [1,100] ("+str(currentNbOfTest)+","+str(maxNbOfTest)+") :"
   #text="Entrer un int [1,100] (%d,%d): " % (currentNbOfTest, maxNbOfTest)
   #text= "Enter an int [1,100] ({},{}): ".format(currentNbOfTest, maxNbOfTest)
   text= f"Enter an int [1,100] ({currentNbOfTest},{maxNbOfTest}): "
   resp=input(text)
   resp=int(resp)
   currentNbOfTest += 1
   if resp < secret:
      print ("Too small !")
   elif resp > secret:
      print ("Too big !")
   else:
      print ("Bingo !")
      break
else:  #or:   if currentNbOfTest > maxNbOfTest:
   print ("Sorry, the secret number was: ", secret)
```

4. Write a Python script that defines a small French/English dictionary (5 or 6 words will be enough). The user of the script will be prompted for a French word and the script will display its corresponding translation in English.
   If the word is not present in the dictionary, the script will print all the words it is able to translate.
   Note: the script should be case insensitive.

```
french=("Mer","Ville","Voiture","Ciel","Couleur")
english=("Sea","Town","Car","Sky","Color")
fr_en=dict(zip(french, english))
# or more directly:
# fr_en={"Mer":"Sea","Ville":"Town","Voiture":"Car","Ciel":"Sky","Couleur":"Color"}
while True:
   answerOrg=input("Enter a french word (or 'Stop'): ")
   answer=answerOrg.lower().capitalize()
   if answer == "Stop":
      print("Bye!")
      break
   if answer in fr_en:
      print("Translation of {} is {}".format(answerOrg, fr_en[answer]))
   else:
      print("No entry for the word: {} in my dict".format(answerOrg))
      print("But I can translate:")
      for fr in sorted(fr_en.keys()):
         print(fr,end=" ")
      print()
```

5. Write a pythons scripts that, given a list of words (a list of str), creates a dictionary where the keys do correspond to the word lengths and the values a list of the words with the corresponding length.
   For instance, if the list is:

   words=["He", "does", "confess", "he", "feels", "himself", "distracted", "but", "from", "what", "cause", "he", "will", "by", "no", "means", "speak"]

   The dict should be:

   ```
   {   2:['He', 'he', 'by', 'no'],
       3:['but'],
       4:['does', 'from', 'what', 'will'],
       5:['feels', 'cause', 'means', 'speak'],
       7:['confess', 'himself'],
       10:['distracted']}
   ```

```
words=["He", "does", "confess", "he", "feels", "himself","distracted", "but", "from", "what", "cause",
"he", "will", "by","no", "means", "speak"]

sizeDict={}

for word in words:
   wordSize=len(word)
   if wordSize in sizeDict:
      if word not in sizeDict[wordSize]:
         sizeDict[wordSize].append(word)
   else:
      sizeDict[wordSize]=[word]

for sz in sizeDict:
   print(f"{sz:3d} --> {sizeDict[sz]}")
```

6. Write a function that do test that the argument it receives is a prime number.
   A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself.
   This function will take as argument the number to test and it will return a Boolean result: True is the number is primer, False if not.
   For instance:
   ```
   isPrime(5) --> True
   isPrime(12) --> False
   ```

   Note: If a number n is not a prime, it can be factored into two factors a and b: n = a*b
   If both a and b were greater than the square root of n, a*b would be greater than n. So at least one of those factors must be less or equal to the square root of n**: to check if n is prime, we only need to test for factors less than or equal to the square root of n**.

```
def isPrime(nb):
    if not isinstance(nb, int):
        return None
    nb=abs(nb)
    if nb==0:
        return False
    if nb==1 or nb==2:
        return True
    import math
    for v in range(2, int(math.sqrt(nb))+1):
        if nb % v == 0:
            return False
    return True

# A few numbers to test:
val=[0,1,2,3,4,5,6,7,8,9,10,23,25,49]

for ix in val:
    if isPrime(ix):
        print (ix, "is a prime number")
    else:
        print (ix, "is NOT a prime number")
```

7. The Fibonacci numbers are defined recursively by the following equations:

    Fn=Fn-1 + Fn-2

    F1=1

    F0=0

   Here are the first elements in the Fibonacci sequence:

    0,1,1,2,3,5,8,13,21,34, ...

   Write a function that returns the n-th Fibonacci number (n being a positive integral argument received by the function).

   You can implement this function in 3 different ways:
   - Using a looping technique
   - Using recursion

```
def fibo_v1(n):
    if n==0: return 0
    if n==1: return 1
    n_1=1
    n_2=0
    for __ in range(2,n+1):
        n=n_1+n_2
        n_2=n_1
        n_1=n
    return n

def fibo_v2(n):
    if n==0: return 0
    if n==1: return 1
    return fibo_v2(n-1) + fibo_v2(n-2)

# First 25 elements in the Fibonacci sequence:
for val in range(25):
    print ("{:3d} ---> {:6d}".format(val, fibo_v1(val)))
```

8. Write a Python script that do define a function `addList()` that is able to add a list of numbers to another list of numbers:
   `addList([3,4,5] , [6,8,9,3,5])` should return *`[9,12,14,3,5]`*

   If the two lists are not of the same size, an optional argument will allow you to indicate which value should be added to the elements that have no counterpart in the other list:
   `addList([3,4,5] , [6,8,9,3,5], 5)` should return *`[9,12,14,8,10]`*

```
# Version 1:
def addList_v1(li1, li2, default=0):
    res=[]
    if len(li1) < len(li2):
        for i in range(len(li1)):
            res.append(li1[i]+li2[i])
        for i in range(len(li1), len(li2)):
            res.append(li2[i]+default)
    else:
        for i in range(len(li2)):
            res.append(li1[i]+li2[i])
        for i in range(len(li2), len(li1)):
            res.append(li1[i]+default)
    return res

# Version 2
def addList_v2(li1, li2, default=0):
    liloc1=li1[:]
    liloc2=li2[:]
    if len(li1) < len(li2):
        liloc1.extend([default]* (len(li2)-len(li1)) )
    else:
        liloc2.extend([default]* (len(li1)-len(li2)) )

    return [liloc1[i]+liloc2[i] for i in range(len(liloc1))]
    # or:
    # return map(lambda a,b: a+b, liloc1, liloc2)

l1=[1,2,3]
l2=[3,4,5,10,23]
l3=addList_v1(l1,l2)
print (l3) # [4,6,8,10,23]
l3=addList_v2(l1,l2)
print (l3) # [4,6,8,10,23]

l1=[1,2,3,10,23]
l2=[3,4,5]
l3=addList_v1(l1,l2)
print (l3) # [4,6,8,10,23]
l3=addList_v2(l1,l2)
print (l3) # [4,6,8,10,23]

l1=[1,2,3]
l2=[3,4,5,10,23]
l3=addList_v1(l1,l2,10)
print (l3) # [4,6,8,20,33]
l3=addList_v2(l1,l2,10)
print (l3) # [4,6,8,10,23]
```

9. Modify exercise 4: a function should now be defined that given a French word returns its translation in English or raise an exception if no translation is available.

```
fr_en={"Mer":"Sea", "Ville":"Town", "Voiture":"Car", "Ciel":"Sky", "Couleur":"Color"}

def translate_fr2en(frWord):
    frWord=frWord.lower().capitalize()
    if frWord in fr_en:
        return fr_en[frWord]
    else:
        raise Exception("No translation found!")


while True:
    answer=input("Enter a french word (or 'stop'): ")
    if answer == "stop":
        print ("Bye!")
        break
    try:
        enWord=translate_fr2en(answer)
        print ("Translation of {} is {}".format(answer, enWord))
    except Exception as e:
        print ("No entry for the word: {} in my dict".format(answer))
        print ("Error message: {}".format(e))
        print ("But I can translate:")
        for fr in sorted(fr_en.keys()):
            print(fr,end=' ')
        print()
```

10. Write a Stack class using a composition mechanism (the Stack class will use internally the List class).
    The Stack class should offer the following facilities:
    - a push() method to add an element on top of the Stack
    - a pop() method to get and remove the element on top of the Stack
    - a peek() method to get the element on top of the Stack (without removing it)
    - a way to test if it is empty or not,
    - a way to determine its size (its current number of elements),
    - a way to test that a Stack is equal to another
    - a way to print easily the content of the Stack
    - a way to test if an element belongs to the stack or not

```
class StackFullError(Exception):
   def __init__(self, message):
      Exception.__init__(self,message)
   # or simply:
   # pass
class StackEmptyError(Exception):
   def __init__(self, message):
      Exception.__init__(self,message)
   # or simply:
   # pass
class StackSizeError(Exception):
   def __init__(self, message):
      Exception.__init__(self,message)
   # or simply:
   # pass
class Stack(object):
   # The constructor:
   def __init__(self, size):
      '''
      The constructor expect to receive as argument an int: the maximum
      size of the stack.
      '''
      self.content=[]     # a list to hold the elements
      if isinstance(size, int):
         if size > 0:
            self.__maxSize=size   # the maximum size of the stack
         else:
            raise StackSizeError("The size of the stack should be > 0 !")
      else:
         raise StackSizeError("The size of the stack should be an int !")

   def push(self, arg):
      '''
      Push add the argument it receives to the stack, if the stack is not full
      '''
      if not self.isFull():
         self.content.append(arg)
      else:
         raise StackFullError("Stack full")
   def pop(self):
      '''
      Pop returns and removes the element currently on top of the stack ...
      ... if the stack is not empty!
      '''
      if not self.isEmpty():
            return self.content.pop()
        else:
            raise StackEmptyError()
    def peek(self):
        '''
        peek returns the element currently on top of the stack ...
        ... if the stack is not empty!
        '''
        if not self.isEmpty():
            return self.content[-1]
        else:
            raise StackEmptyError("The stack is empty !")
    def isEmpty(self):
        '''
        isEmpty returns True is the Stack is empty, False otherwise!
        '''
        return  len(self.content) == 0
    def isFull(self):
        '''
        isFull returns True is the Stack is full, False otherwise!
        '''
        return  len(self.content) == self.__maxSize
    def __len__(self):
        '''
        __len__ is invoked by the builtin len(), it should return the current
        size of the Stack
        '''
        return  len(self.content)
    def __str__(self):
        '''
        __str__ is invoked by the builtin str(), it should return a string
        representation of the Stack
        '''
        return  "({}/{}) {}".format(len(self.content), self.__maxSize, self.content)
    def __eq__(self, param):
        '''
        __eq__ is invoked by ==, it should return a boolean
        '''
        if self.__maxSize != param.__maxSize:
            return False
        return self.content == param.content
    def __contains__(self, val):
        return val in self.content


if __name__=="__main__":
    try:
        s=Stack(5)      # A stack with a maximum size of 5 elements
        print("s", s)    # (0/5) []
        s.push(10)
        # Stack.push(s,10)  # Not recommended but legal !
        s.push(12)
        if 10 in s:
            print("10 is in s")
        print("s", s)    # (2/5) [10,12]
        t=s.pop()
        print("t", t)    # t 12
        print("s", s)    # (1/5) [10]
        t=s.peek()
        print("t", t)    # t 10
        print("s", s)    # (1/5) [10]
        s.push(100)
        s.push(33)
        print("s", s)    # (3/5) [10,100,33]
        print("Size", len(s))  # Size 3
    except Exception as e:
        print("Problem:", e)
```

11. Now, implement the Stack class using an inheritance mechanism (the Stack class will inherit from the List class).
    Compare this version of the Stack class with the previous one.

```
class StackFullError(Exception):
    def __init__(self, message):
        Exception.__init__(self,message)
    # or simply:
    # pass
class StackEmptyError(Exception):
    def __init__(self, message):
        Exception.__init__(self,message)
    # or simply:
    # pass
class StackSizeError(Exception):
    def __init__(self, message):
        Exception.__init__(self,message)
    # or simply:
    # pass
class Stack(list):
    # The constructor:
    def __init__(self, size):
        '''
        The constructor expect to receive as argument an int: the maximum
        size of the stack.
        '''
        list.__init__(self) # or super().__init__()
        if isinstance(size, int):
            if size > 0:
                self.__maxSize=size   # the maximum size of the stack
            else:
                raise StackSizeError("The size of the stack should be > 0 !")
        else:
            raise StackSizeError("The size of the stack should be an int !")

    def push(self, arg):
        '''
        Push add the argument it receives to the stack, if the stack is not full
        '''
        if not self.isFull():
            self.append(arg)
        else:
            raise StackFullError("Stack full")

    def pop(self):
        '''
        Pop returns and removes the element currently on top of the stack ...
        ... if the stack is not empty!
        '''
        if not self.isEmpty():
            return list.pop(self)
        else:
            raise StackEmptyError()

    def peek(self):
        '''
        peek returns the element currently on top of the stack ...
        ... if the stack is not empty!
        '''
        if not self.isEmpty():
            return self[-1]
        else:
            raise StackEmptyError("The stack is empty !")
    def isEmpty(self):
        '''
        isEmpty returns True is the Stack is empty, False otherwise!
        '''
        return  len(self) == 0
    def isFull(self):
        '''
        isFull returns True is the Stack is full, False otherwise!
        '''
        return  len(self) == self.__maxSize
    def __str__(self):
        '''
        __str__ is invoked by the builtin str(), it should return a string
        representation of the Stack
        '''
        return  "({}/{}) {}".format(len(self), self.__maxSize, list.__str__(self))
    def __eq__(self, param):
        '''
        __eq__ is invoked by ==, it should return a boolean
        '''
        if self.__maxSize != param.__maxSize:
            return False
        return list.__eq__(self,param) # or super().__eq__(param)


if __name__=="__main__":
    try:
        s=Stack(5)      # A stack with a maximum size of 5 elements
        print("s", s)    # (0/5) []
        s.push(10)
        # Stack.push(s,10)  # Not recommended but legal !
        s.push(12)
        if 10 in s:
            print("10 is in s")
        print("s", s)    # (2/5) [10,12]
        t=s.pop()
        print("t", t)    # t 12
        print("s", s)    # (1/5) [10]
        t=s.peek()
        print("t", t)    # t 10
        print("s", s)    # (1/5) [10]
        s.push(100)
        s.push(33)
        print("s", s)    # (3/5) [10,100,33]
        print("Size", len(s))  # Size 3
    except Exception as e:
        print("Problem:", e)
```

12. Write a class Date to represent a date (day, month and year).
    A class Time to represent a time (hour, minute seconds).
    These classes should provide methods to create, initialize, modify, display, compare, …
    dates and times respectively.
    Then create a class DateTime that inherits from both Date and Time.

```
class Date(object):
    def __init__(self, p1, p2, p3):
        if 1<p1<31:
            self.__day=p1
        else:
            self.__day=1
        if 1<p2<12:
            self.__month=p2
        else:
            self.__month=1
        if 1900<p3<2100:
            self.__year=p3
        else:
            self.__year=2016

    def setDay(self, param):
        if 1<param<31:
            self.__day=param
        else:
            raise Exception("Wrong day value")

    def getDay(self):
        return self.__day

    day=property(getDay, setDay)

    def setMonth(self, param):
        if 1<param<12:
            self.__month=param
        else:
            raise Exception("Wrong month value")

    def getMonth(self):
        return self.__month

    month=property(getMonth, setMonth)

    def setYear(self, param):
        if 1900<param<2100:
            self.__year=param
        else:
            raise Exception("Wrong year value")

    def getYear(self):
        return self.__year

    year=property(getYear, setYear)

    def __eq__(self, param):
        return self.__day == param.__day and self.__month == param.__month and self.__year ==
param.__year

    def __str__(self):
        return  "{:d}/{:d}/{:d}".format(self.__day, self.__month, self.__year)

class Time(object):

    # Definition of a constructor:
    def __init__(self, p1, p2, p3):
        if 0<=p1<=23:
            self.__hour=p1
        else:
            self.__hour=0
        if 0<=p2<=59:
            self.__minute=p2  # A instance attribute
        else:
            self.__minute=0
        if 0<=p3<=59:
            self.__second=p3
        else:
            self.__second=0

    def setHour(self, param):
        if 0<=param<=23:
            self.__hour=param
        else:
            raise Exception("Wrong hour value")

    def getHour(self):
        return self.__hour

    hour=property(getHour, setHour)

    def setMinute(self, param):
        if 0<=param<=59:
            self.__minute=param
        else:
            raise Exception("Wrong minute value")

    def getMinute(self):
        return self.__minute

    minute=property(getMinute, setMinute)
    def setSecond(self, param):
        if 0<=param<=59:
            self.__second=param
        else:
            raise Exception("Wrong second value")

    def getSecond(self):
        return self.__second

    second=property(getSecond, setSecond)

    def __eq__(self, param):
        return self.__hour == param.__hour and self.__minute == param.__minute and self.__second ==
param.__second

    def __str__(self):
        return  "{:d}:{:d}:{:d}".format(self.__hour, self.__minute, self.__second)

    def __repr__(self):
        return  "{:d}:{:d}:{:d}".format(self.__hour, self.__minute, self.__second)
```

```
class DateTime(Date, Time):
   def __init__(self, p1,p2,p3,p4,p5,p6):
       Date.__init__(self, p1,p2,p3)
       Time.__init__(self, p4,p5,p6)

   def __str__(self):
       return  "{:d}/{:d}/{:d} {:d}:{:d}:{:d}".format(self.day, self.month, self.year,self.hour, self.minute,
self.second)

dt1=DateTime(12,3,2003, 23,4,55)

print("DT1 month", dt1.month)
print("DT1 second", dt1.second)
dt1.hour=22 # Inherited from Time
dt1.day=24  # Inherited from Date
print("DT1", dt1)
```

13. Given a file data.txt with the following content:

```
City  Temperature
Gland 23.4
Nyon 23.6
Geneva 22.8
Lausanne 24.4
Versoix 23.4
Gland 22.4
Nyon 21.8
Geneva 22.8
Lausanne 28.4
Gland 22.9
Nyon 22.8
Geneva 23.2
Lausanne 23.5
Versoix 22.9
….
```

Read the file "data.txt" and use it to create a dictionary within which the key will be a city name and the value associated with it a list of temperature.

Write a Python script that does answer the following questions:

- How many measures have been recorded for a given city (Geneva for instance) ?
- What are the temperatures recorded for a given city (Lausanne for instance)?
- What is the average of the temperature for a given city (Nyon for instance)?
- How many cities are present?

Create a summary file with a format like the following:

```
Nyon: [23.6, 21.8, 22.8] Average:22.73
Gland: [23.4, 22.4, 22.9] Average:22.90
…
```

```
class TempProcess(object):
   def __init__(self, fileName):
      self.__content={}
      self.readFromFile(fileName)
   def readFromFile(self, fileName):
      try:
         first=True
         with open(fileName) as f:

            for line in f:
               if first: # To skip the first line
                  first=False
                  continue
               result=line.split()
               if len(result) != 2:
                  continue
               city=result[0].strip().lower()
               try:
                  temp=float(result[1].strip())
               except ValueError:
                  print("Wrong temperature format")
                  continue
               if city in self.__content:
                  self.__content[city].append(temp)
               else:
                  self.__content[city]=[temp]
      except Exception as e:
         print("Problem with the file", fileName)
         print(e)
   def getTemperatures(self, city):
      cityl=city.lower()
      if cityl in self.__content:
         return self.__content[cityl]
      else:
         print("Unknown city", city)
         return None
   def getAverage(self, city):
      cityl=city.lower()
      if cityl in self.__content:
         return sum(self.__content[cityl])/len(self.__content[cityl])
      else:
         print("Unknown city", city)
         return None
   def getNumberOfMesures(self, city):
      cityl=city.lower()
      if cityl in self.__content:
         return len(self.__content[cityl])
      else:
         print("Unknown city", city)
         return None
   def getNumberOfCities(self):
      return len(self.__content)
   def saveSummary(self, fileName):
      try:
         with open(fileName,"w") as f:
            for city in self.__content:
               f.write("{}: {} Average:{:.2f}\n".format(city.capitalize(), self.__content[city],
self.getAverage(city)))
      except Exception as e:
        print("Problem with the file", fileName)
        print(e)

tp=TempProcess("data.txt")
print("Geneva appears {} time in the file".format(tp.getNumberOfMesures("Geneva")))
print("The average of the temperature in Nyon is {:.2f}".format(tp.getAverage("Nyon")))
print("{} different cities are present in the file".format(tp.getNumberOfCities()))
print("The temperatures recorded in Lausanne are {}".format(tp.getTemperatures("Lausanne")))
tp.saveSummary("data_summary.txt")
```