ids-pdl06-hwk.ipynb: This Jupyter notebook is provided by Joachim Vogt for the *Python Data Lab* of the module *Introduction to Data Science* offered in Fall 2022 at Jacobs University Bremen. Module instructors are Hilke Brockmann, Adalbert Wilhelm, and Joachim Vogt. Jupyter notebooks and other learning resources are available from a dedicated *module platform*.

Homework: Control flow statements

The homework assignments in this notebook supplement the tutorial *Control flow statements*.

- Solve the assignments according to the instructions.
- Upload the completed notebook to the module platform.
- Do not forget to enter your name in the markdown cell below.

The homework set carries a total of 20 points. Square brackets in the assignment titles specify individual point contributions.

Name: Hamza Rehman

Preparation

The text file `pdl06sol_forloop_Nmax=15.txt` is expected to reside in the working directory. Identify the file on the module platform and upload it to the same folder as this Jupyter notebook.

Assignment: Conditional statements [5]

Using three nested `if-else` blocks, provide a piece of Python code to check if a given year is a leap year. To prepare for this assignment, review the logic and the syntax of Python module operation `%`, and consider the following rules. Display the result by means of another `if-else` construction.

1. Years that are not divisible by 4 are not leap years.
2. Years that are divisible by 4 but not by 100 are leap years.
3. Years that are divisible by 100 but not by 400 are not leap years.
4. Years that are divisible by 400 are leap years.

Complete the code cell below according to the instructions included as comments. More concretely, replace the `pass` statements with appropriate instructions, and the instances of `(42)` with correct boolean expressions.

In [1]:
```python
### Define a list of years to test the nested if-else constructions.
YearList = [800,1492,1776,1789,1900,1945,2000]

### Initialize boolean variable LeapYear.
LeapYear = False

### Loop over the elements of the list of years.
for Year in YearList:

### if-else (1): Year is not divisible by 4.
    if (Year % 4 !=0):
        LeapYear = False
    else:

###.... if-else (2): Year is divisible by 4 but not by 100.
        if (Year % 100 !=0):
            LeapYear = True
        else:

###........ if (3): Year is divisible by 100 but not by 400.
            if (Year % 400 !=0):
                LeapYear = False

###........ else 4: Year is divisible by 400.
            else:
                LeapYear = True

### Display the result by means of another if-else construction.
    if not LeapYear:
        print('Year {:4d} was not a leap year.'.format(Year))
    else:
        print('Year {:4d} was a leap year.'.format(Year))
```
```
Year  800 was a leap year.
Year 1492 was a leap year.
Year 1776 was a leap year.
Year 1789 was not a leap year.
Year 1900 was not a leap year.
Year 1945 was not a leap year.
Year 2000 was a leap year.
```

Assignment: `for` loop [3]

Loops can be used to evaluate sequences or series in mathematics. As an example for a sequence, we consider the formula for *compound interest*:

$$C_n \ = \ \left(1+\frac{1}{n+1}\right)^{n+1} \ , \ n=0,1,2,3,\ldots$$

As an example for a series, we consider the Taylor expansion of the exponential function at zero:

$$E_n \ = \ \sum_{k=0}^{n} \frac{1}{k!} \ , \ n=0,1,2,3,\ldots$$

Note that both $\{C_n\}_{n=0,1,2,\ldots}$ and $\{E_n\}_{n=0,1,2,\ldots}$ converge toward $e = 2.7182818\ldots$ as $n \to \infty$.

In the following code cell, add a `for` loop to compute and display $\{C_n\}_{n=0,1,2,\ldots,N_{\max}}$ and $\{E_n\}_{n=0,1,2,\ldots,N_{\max}}$ for $N_{\max} = 15$. Compare your results with the content of the file `pdl06sol_forloop_Nmax=15.txt`, see below.

In [2]:
```python
### In the formula for En, n! can be computed by means of factorial(n) from the module math.
from math import factorial

### Set maximum Nmax of iteration counter n.
Nmax = 15

### Initialize En.
En = 0

### Create for loop to compute and print Cn and En for n=0,1,2,...,Nmax.
for n in range(0,Nmax+1):
    En= En + 1/ factorial(n)
    C = (1 + (1 / (n +1))) ** (n + 1)
    print("Iteration {:2d}: C = {:.12f}, E = {:.12f}".format(n,C,En))
```
```
Iteration  0: C = 2.000000000000, E = 1.000000000000
Iteration  1: C = 2.250000000000, E = 2.000000000000
Iteration  2: C = 2.370370370370, E = 2.500000000000
Iteration  3: C = 2.441406250000, E = 2.666666666667
Iteration  4: C = 2.488320000000, E = 2.708333333333
Iteration  5: C = 2.521626371742, E = 2.716666666667
Iteration  6: C = 2.546499697041, E = 2.718055555556
Iteration  7: C = 2.565784513950, E = 2.718253968254
Iteration  8: C = 2.581174791713, E = 2.718278769841
Iteration  9: C = 2.593742460100, E = 2.718281525573
Iteration 10: C = 2.604199011898, E = 2.718281801146
Iteration 11: C = 2.613035290225, E = 2.718281826198
Iteration 12: C = 2.620600887886, E = 2.718281828286
Iteration 13: C = 2.627151556301, E = 2.718281828447
Iteration 14: C = 2.632878717728, E = 2.718281828458
Iteration 15: C = 2.637928497367, E = 2.718281828459
```

See the file `pdl06sol_forloop_Nmax=15.txt` for the correct results.

In [3]:
```python
f = open('pdl06sol_forloop_Nmax=15.txt','r')
smpout = f.read()
f.close()
print(smpout)
```
```
Iteration  0:  C = 2.000000000000 ,  E = 1.000000000000
Iteration  1:  C = 2.250000000000 ,  E = 2.000000000000
Iteration  2:  C = 2.370370370370 ,  E = 2.500000000000
Iteration  3:  C = 2.441406250000 ,  E = 2.666666666667
Iteration  4:  C = 2.488320000000 ,  E = 2.708333333333
Iteration  5:  C = 2.521626371742 ,  E = 2.716666666667
Iteration  6:  C = 2.546499697041 ,  E = 2.718055555556
Iteration  7:  C = 2.565784513950 ,  E = 2.718253968254
Iteration  8:  C = 2.581174791713 ,  E = 2.718278769841
Iteration  9:  C = 2.593742460100 ,  E = 2.718281525573
Iteration 10:  C = 2.604199011898 ,  E = 2.718281801146
Iteration 11:  C = 2.613035290225 ,  E = 2.718281826198
Iteration 12:  C = 2.620600887886 ,  E = 2.718281828286
Iteration 13:  C = 2.627151556301 ,  E = 2.718281828447
Iteration 14:  C = 2.632878717728 ,  E = 2.718281828458
Iteration 15:  C = 2.637928497367 ,  E = 2.718281828459
```

Assignment: `while` loop [6]

In the cell below, you find incomplete Python code to play one round of a simplified version of Black Jack.

- The finite size of the card deck is neglected so that the probability of drawing a particular type of card does not change throughout the game.
- An ace always counts as 1.
- As usual, number cards (2-10) count as their number and face cards (Jacks, Queens, Kings) count as 10.
- You receive two cards. Then you can ask for an additional card as often as you like but when the total exceeds 21 (an outcome called *bust*) you loose.
- After you have finished asking for additional cards, the dealer's hand is collected. Here cards are added as long as the total of the dealer's hand is smaller than 17. If the dealer busts but your total is 21 or below, you win.
- If both you and the dealer arrive at totals of 21 or less (i.e., nobody busts), the hand with the largest total wins. If there is a tie (i.e., both hands have the same total), the dealer wins.

Study the incomplete piece of code in the following cell. Following the example of the first `while` loop which completes the list `YourHand` and computes `YourTotal`, replace the `pass` statement with a second `while` loop that completes the list `DealersHand` and computes `DealersTotal`. In the `if-else` construction used to compare `YourTotal` and `DealersTotal`, replace instances of the number `(42)` with appropriate boolean expressions.

In [6]:
```python
### Import randint to simulate random draws of cards.
from numpy.random import randint

### Define lists of cards and their values.
ListOfCards = ['Ace','Two','Three','Four','Five','Six','Seven','Eight','Nine','Ten','Jack','Queen','King']
ListOfValues = [1,2,3,4,5,6,7,8,9,10,10,10,10]

### Your first card is drawn.
Ind = randint(13)
YourHand = [ListOfCards[Ind]]
YourTotal = ListOfValues[Ind]
OneMoreCard = 'Y'
### Now you are asked for additional cards.
while OneMoreCard.upper()=='Y':
    Ind = randint(13)
    YourHand.append(ListOfCards[Ind])
    print('Your hand      : ',YourHand)
    YourTotal = YourTotal + ListOfValues[Ind]
    if YourTotal>21:
        print("Your total     : ",YourTotal)
        print("The dealer wins because your total exceeds 21.")
        break
    else:
        OneMoreCard = input('One more card (answer Y or N) : ')
else:
### The dealer's cards are drawn.
    print()
    Ind = randint(13)
    DealersHand = [ListOfCards[Ind]]
    DealersTotal = ListOfValues[Ind]
while DealersTotal < 17:
    Ind = randint(13)
    DealersHand.append(ListOfCards[Ind])
    print("Dealer's Hand      : ",DealersHand)
    DealersTotal = DealersTotal + ListOfValues[Ind]
    if DealersTotal > 21:
        print("Dealer's Total     : ", DealersTotal)
        print("The dealer wins because your total exceeds 21.")
        break

### Compare the totals to decide who wins.
if (YourTotal <=21) and (DealersTotal <=21):
    print("Your total     : ",YourTotal)
    print("Dealer's total : ",DealersTotal)
    if (YourTotal > DealersTotal):
        print("You win because your total is larger than the dealer's total.")
    else:
        print("The dealer wins because your total is not larger than the dealer's total.")
```
```
Your hand      :  ['Eight', 'Seven']
One more card (answer Y or N) : N

Dealer's Hand      :  ['Eight', 'Three']
Dealer's Hand      :  ['Eight', 'Three', 'Six']
Your total     :  15
Dealer's total :  17
The dealer wins because your total is not larger than the dealer's total.
```

Assignment: List comprehensions [6]

For each of the following control structures, find a list comprehension that yields the same result.

In [9]:
```python
### First control structure.
base = 0.5
cs1 = []
for k in range(9):
    cs1.append(base**k)
### First list comprehension, equivalent to the first control structure.
lc1 = [base**k for k in range(9)]

### Comparison of control structure and list comprehension.
print('*** First example (geometric sequence) ***')
print('Control structure  : ',cs1)
print('List comprehension : ',lc1)
print()

### Second control structure.
Alphabet = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
Vowels = list('AEIOU')
cs2 = []
for char in Alphabet:
    if char not in Vowels:
        cs2.append(char)
### Second list comprehension, equivalent to the second control structure.
lc2 = [char for char in Alphabet if char not in Vowels]
### Comparison of control structure and list comprehension.
print('*** Second example (consonants) ***')
print('Control structure  : ',cs2)
print('List comprehension : ',lc2)
print()

### Third control structure.
from scipy.special import comb
cs3 = []
for N in range(5):
    for k in range(N+1):
        cs3.append((N,k,comb(N,k,exact=True)))
### Third list comprehension, equivalent to the third control structure.
lc3 = [(N,k,comb(N,k,exact=True)) for N in range(5) for k in range(N+1)]
### Comparison of control structure and list comprehension.
print('*** Third example (binomial coefficients) ***')
print('Control structure  : ',cs3)
print('List comprehension : ',lc3)
print()
```
```
*** First example (geometric sequence) ***
Control structure  :  [1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125, 0.00390625]
List comprehension :  [1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125, 0.00390625]

*** Second example (consonants) ***
Control structure  :  ['B', 'C', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z']
List comprehension :  ['B', 'C', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z']

*** Third example (binomial coefficients) ***
Control structure  :  [(0, 0, 1), (1, 0, 1), (1, 1, 1), (2, 0, 1), (2, 1, 2), (2, 2, 1), (3, 0, 1), (3, 1, 3), (3, 2, 3), (3, 3, 1), (4, 0, 1), (4, 1, 4), (4, 2, 6), (4, 3, 4), (4, 4, 1)]
List comprehension :  [(0, 0, 1), (1, 0, 1), (1, 1, 1), (2, 0, 1), (2, 1, 2), (2, 2, 1), (3, 0, 1), (3, 1, 3), (3, 2, 3), (3, 3, 1), (4, 0, 1), (4, 1, 4), (4, 2, 6), (4, 3, 4), (4, 4, 1)]
```
