CE376 - Programming in Python [Practical File]

18DCE115 Kashyap Shah

Practical 1

Aim: Create a program that asks the user to enter their name and their age. Printout a message addressed to them that tells them the year that they will turn 100 years old.

In [3]:
```
from datetime import datetime

# asks the user to enter their names and ages

name = input('Enter your name: ')
age = input('Enter your age:  ')
currentYear = datetime.now().year

# print a message to show the year that they will be 100 years old

year = currentYear - int(age) + 100
message = name + " will be 100 years old in " + str(year) + "\n"
print(message)
```

```
Enter your name: Kashyap
Enter your age: 20
Kashyap will be 100 years old in 2101
```

Practical 2.1

Aim: Ask the user for a number. Depending on whether the number is even or 2 odd, print out an appropriate message to the user. Hint: how does an even / odd number react differently when divided by 2?

In [6]:
```
num = int(input('Enter a number '))
check = num % 2
if check==0:
    print(str(num) + ' is Even ')
else:
    print(str(num) + ' is odd ')
```

```
Enter a number 1234
1234 is Even
```

Practical 2.2

Aim: Take a list, say for example this one: a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89], and write a program that prints out all the elements of the list that are less than 5.

In [7]:
```
list = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
for i in list:
    if list[i]<5:
        print(list[i])
    else:
        break
```

```
1
1
2
3
```

Practical 3.1

Aim: Create a program that asks the user for a number and then prints out a list of all the divisors of that number. (If you don't know what a divisor is, it is a number that divides evenly into another number. For example, 13 is a divisor of 26 because 26 / 13 has no remainder.)

In [8]:
```
num=int(input('Enter a number '))
i=1
while i <= num:
    if (num%i==0):
        print(i),
    i+=1
```

```
Enter a number  26
1
2
13
26
```

Practical 3.2

Aim: Take two lists, say for example these two: a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89] b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes.

In [2]:
```
global str
list1 = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
list2 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

# by using inbuilt set() which is used convert any of the iterable to sequence of it
# elements with distinct elements, commonly called Set.
c = list(set(list1) & set(list2))
print(c)
```

```
[1, 2, 3, 5, 8, 13]
```

Practical 3.3

Aim :Ask the user for a string and print out whether this string is a palindrome or not. (A palindrome is a string that reads the same forwards and backwards.)

In this particular example, the slice statement [::-1] means start at the end of the string and end at position 0, move with the step -1, negative one, which means one step backwards.

In [4]:
```
string=input('Enter a string ')
rev=string[::-1]

print("\nActual string : "+ string)
print('Reversed string : "+ rev)

if string==rev:
    print("\nPalindrome")
else:
    print("\nNot palindrome")
```

```
Enter a string Hello
```

```
Actual string :  Hello
Reversed string : olleH

Not palindrome
```

Practical 4.1

Aim : Let's say I give you a list saved in a variable: a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]. Write one line of Python that takes this list and makes a new list that has only the even elements of this list in it.

In [5]:
```
a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
print([num for num in a if num%2==0])
```

```
[4, 16, 36, 64, 100]
```

Practical 4.2

Aim : Make a two-player Rock-Paper-Scissors game. (Hint: Ask for player plays (using input), compare them, print out a message of congratulations to the winner, and ask if the players want to start a new game) Remember the rules: Rock beats scissors, Scissors beats paper, Paper beats rock.

In [2]:
```
from random import randint

#create a list of play options
t = ["Rock", "Paper", "Scissors"]

#assign a random play to the computer
computer = t[randint(0,2)]

#set player to False
player = False

while player == False:
#set player to True
    player = input("Enter Your Move (Rock, Paper, Scissors or Exit): ")
    if player == computer:
        print("Computer: ", computer)
        print("Result: Tie!")
    elif player == "Rock":
        if computer == "Paper":
            print("Computer: ", computer)
            print("Result: You lose!", computer, "covers", player)
        else:
            print("Computer: ", computer)
            print("Result: You win!", player, "smashes", computer)
    elif player == "Paper":
        if computer == "Scissors":
            print("Computer: ", computer)
            print("Result: You lose!", computer, "cut",  player)
        else:
            print("Computer: ", computer)
            print("Result: You win!", player, "covers", computer)
    elif player == "Scissors":
        if computer == "Rock":
            print("Computer: ", computer)
            print("Result: You lose!", computer, "smashes", player)
        else:
            print("Computer: ", computer)
            print("Result: You win!", player, "cut", computer)
```

```
    elif player == "Exit":
        break
    else:
        print("That's not a valid play. Check your spelling!")
    #player was set to True, but we want it to be False so the loop continues
    player = False
    computer = t[randint(0,2)]
```

```
Enter Your Move (Rock, Paper, Scissors or Exit): Rock
Computer:  Rock
Result: Tie!
Enter Your Move (Rock, Paper, Scissors or Exit): Paper
Computer:  Scissors
Result: You lose! Scissors cut Paper
Enter Your Move (Rock, Paper, Scissors or Exit): Exit
```

Practical 4.3

Aim : Generate a random number between 1 and 9 (including 1 and 9). Ask the user to guess the number, then tell them whether they guessed too low, too high, or exactly right. (Hint: remember to use the user input lessons from the very first practical)

In [8]:
```
import random
usr=int(input("Enter a Number (0 to 9): "))
random=random.randint(0,9)
print("Random Number: ",random)
if usr>random:
    print("You Guessed High.")
elif usr<random:
    print("You Guessed Low.")
else:
    print("Excellet! Exactly Right Guess.")
```

```
Enter a Number (0 to 9): 5
Random Number:  7
You Guessed Low.
```

Practical 5.1

Aim : This week's exercise is going to be revisiting an old exercise (see Practical 3), except require the solution in a different way. Take two lists, say for example these two: a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89] b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes. Write this in one line of Python using at least one list comprehension

In [9]:
```
b = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,100,200,300,500]
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,100]

c = [num for num in a if num in b]
unique = set(c)
print(unique)
```

```
{1, 2, 3, 100, 5, 8, 13}
```

Practical 5.2

Aim : Ask the user for a number and determine whether the number is prime or not. (For those who have forgotten, a prime number is a number that has no divisors.). You can (and should!)

use your answer to Practical 2 to help you. Take this opportunity to practice using functions, described below

In [12]:
```
num = int(input('Enter a Number: '))
def prime(num):
    if num>1:
        for i in range(2,num):
            if num%i==0:
                print('It is Not a Prime Number.')
                break
        else:
            print('It is a Prime  Number.')
    else:
        print('It is Not a Prime Number.')

prime(num)
```

```
Enter a Number: 22
It is Not a Prime Number.
```

Practical 5.3

Aim : Write a program that takes a list of numbers (for example, a = [5, 10, 15, 20, 25]) and makes a new list of only the first and last elements of the given list. For practice, write this code inside a function.

In [2]:
```
#Function to return extreme elements from the list
def returnExtremes(oldList):
    newList = []
    newList.append(oldList[0])
    newList.append(oldList[-1])
    return newList

list_a = [5,10,15,20,25]
print(returnExtremes(list_a))
```

```
[5, 25]
```

Practical 6.1

Aim : Write a program that asks the user how many Fibonacci numbers to generate and then generates them. Take this opportunity to think about how you can use functions. Make sure to ask the user to enter the number of numbers in the sequence to generate.

In [4]:
```
def fibo(n):
    a = 0
    b = 1
    c = 0
    fibo_list = []
    fibo_list.append(a)
    fibo_list.append(b)

    if(n == 0):
        print("0")
    elif(n == 1):
        print("0")
    elif(n == 2):
        print(fibo_list)
    else:
        for i in range(3,n+1):
            c = a + b
            a = b
```

```
            b = c
            fibo_list.append(c)
        print(fibo_list)

N = int(input("Enter the limit: "))
print(N,"fibonacci terms are: ")
fibo(N)
```

```
Enter the limit: 10
10 fibonacci terms are:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Practical 6.2

Aim : Write a program (function!) that takes a list and returns a new list that contains all the elements of the first list minus all the duplicates.

In [5]:
```
def uniqueElements(oldList):
    return list(set(oldList))

demo_list = [1,2,2,1,3,5,6,7,3,5,8]
print(uniqueElements(demo_list))
```

```
[1, 2, 3, 5, 6, 7, 8]
```

Practical 6.3

Aim : Write a program (using functions!) that asks the user for a long string function. Containing multiple words. Print back to the user the same string, except with the words in backwards order. For example, say I type the string: My name is Michele Then I would see the string: Michele is name My.

In [7]:
```
def reverseSentence(st):
    return " ".join((st.split(" ")[::-1]))

sentence = str(input("Enter your sentence: "))
print(reverseSentence(sentence))
```

```
Enter your sentence: My Name is Kashyap Shah!
Shah! Kashyap is Name My
```

Practical 7.1

Aim : Write a password generator in Python. Be creative with how you generate passwords - strong passwords have a mix of lowercase letters, uppercase letters, numbers, and symbols. The passwords should be random, generating a new password every time the user asks for a new password. Include your run-time code in a main method.

In [9]:
```
from random import randint
def generatePassword():
    passlen = randint(8,11)
    password = "".join([chr(randint(33,126)) for i in range(passlen)])
    print("Generated Password is ",password)

if __name__=="__main__":
    generatePassword()
```

```
Generated Password is  kA^tJSJ#v
```

Practical 7.2

Aim : Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.

In [10]:
```
import math
class Circle:
    # Constructor
    def __init__(self,radius):
        self.radius = radius
    # Method that returns area
    def calc_area(self):
        print("Area is ",math.pi*self.radius**2)
    # Method that returns perimeter
    def calc_perimeter(self):
        print("Perimeter is ",2*math.pi*self.radius)

circle1 = Circle(5) # create object
circle1.calc_area() # call calc_area()
circle1.calc_perimeter() # call calc_perimeter()
```

```
Area is  78.53981633974483
Perimeter is  31.41592653589793
```

Practical 8.1

Aim : Python supports classes inheriting from other classes. The class being inherited is called the Parent or Superclass, while the class that inherits is called the Child or Subclass. How can we define the order in which the base classes are searched when executing a method?

In [11]:
```
class Parent:
    # Constructor
    def __init__(self,num):
        print("Inside Parent Constructor")
        self.num = num
    # Method to display value in num
    def displayNum(self):
        print("(From Parent) Number is ",self.num)
    # Method to print a message
    def message(self):
        print("Greetings from Parent Class")

class Child(Parent):
    # Constructor
    def __init__(self,num):
        super().__init__(num) # Calling Parent
        print("Inside Child Constructor")
        self.num = num
    # Method to display valie in  num
    def displayNum(self):
        print("(From Child) Number is ",self.num)

obj = Child(5) # create object
obj.displayNum() # calls Child's displayNum()
obj.message()  # calls Parent's message()
```

```
Inside Parent Constructor
Inside Child Constructor
(From Child) Number is  5
Greetings from Parent Class
```

Practical 8.2

Aim : Write a function that takes an ordered list of numbers (a list where the elements are in order from smallest to largest) and another number. The function decides whether or not the given number is inside the list and returns (then prints) an appropriate boolean.

In [12]:
```
def belongsToList(numList, num):
    return num in numList

numList = [1,2,3,4,5,6,7,8]
num = 5
print(belongsToList(numList,num))
```

```
True
```

Practical 8.3

Aim : Given a .txt file that has a list of a bunch of names, count how many of each name there are in the file, and print out the results to the screen.

In [18]:
```
# initialse empty dictionary
name_count = {}
# read from file
with open("names.txt","r") as fp:
    for name in fp:
        try:
            name_count[name] += 1 # if key exists, increment counter by `
        except KeyError: # if key doesn't exist
            name_count[name] = 1 # make a new key and initialise its counter by 1

print(name_count)
```

```
{'kashyap\n': 2, 'john\n': 1, 'dev\n': 1, 'raj\n': 1, 'het\n': 1, 'ram': 1}
```

Practical 9.1

Aim : Develop programs to learn regular expressions using python.

In [19]:
```
import re
# experiment string
txt = "CE376-Programming in Python (Kashyap Shah - 18DCE115)"

# findall() looks for all  occurences
x = re.findall("th", txt)
print("Occurrences of  'th': ",x)
x = re.findall("java",txt)
print("Occurrences of 'java':  ",x)

# search() matches for a string
x = re.search("the",txt)
print("Search for 'the':  ",x)

# split() returns a list based on delimeter
x = re.split("\s",txt)
print("Split using space delimeter: ",x)

# sub() function does replacement
x = re.sub("Python","Java",txt)
print("Replacing 'Python' with 'Java':  ",x)
```

```
Occurrences of 'th':  ['th']
Occurrences of 'java':[]
Search for 'the':  None
Split using space delimeter:  ['CE376-Programming', 'in', 'Python', '(Kashyap', 'Sha
```

```
h', '-', '18DCE115)']
Replacing 'Python' with 'Java': CE376-Programming in Java (Kashyap Shah - 18DCE115)
```

Practical 9.2

Aim : Develop programs for data structure algorithms using python – sorting (Bubble sort and Insertion sort).

In [21]:
```
from random import randint
# function to perform bubble sort
def bubble_sort(A):
    n = len(A)
    for i in range(0,n):
        swapped = False
        for j in range(0,n-i-1):
            if(A[j+1] < A[j]):
                swapped = True
                temp = A[j+1]
                A[j+1] = A[j]
                A[j] = temp
        if(not swapped):
            break

    print("Array Sorted with Bubble Sort: ",A)

# function to perform insertion sort
def insertion_sort(A):
    n = len(A)
    for i in range(1,n):
        key = A[i]
        j = i-1
        while(j>-1 and A[j]>key):
            A[j+1] = A[j]
            j-=1
        A[j+1] = key

    print("Array Sorted with Insertion Sort: ",A)

if __name__=="__main__":
    # make array of random integers
    A = [randint(1,100) for i in range(20)]
    print("Array Before Sort: ",A)
    bubble_sort(A)
    insertion_sort(A)
```

```
Array Before Sort:  [48, 40, 38, 40, 20, 9, 99, 73, 20, 96, 33, 56, 8, 32, 38, 75, 49, 69, 22, 4]
Array Sorted with Bubble Sort:  [4, 8, 9, 20, 20, 22, 32, 33, 38, 38, 40, 40, 48, 49, 56, 69, 73, 75, 96, 99]
Array Sorted with Insertion Sort:  [4, 8, 9, 20, 20, 22, 32, 33, 38, 38, 40, 40, 48, 49, 56, 69, 73, 75, 96, 99]
```

Practical 9.3

Aim : Develop programs to understand working of exception handling and assertions.

In [23]:
```
import sys

randomList=['a',0,2]

for entry in randomList:
    try:
        print("The entry is",entry)
        r=1/int(entry)
```

```
            break
        except:
            print("Oops!",sys.exc_info()[0],"occurred.")
            print("Next entry.")
            print()

print("The reciprocal of",entry,"is",r)
```

```
The entry is a
Oops! <class 'ValueError'> occurred.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError'> occurred.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

Practical 10

Aim : Introduction to Django- Python based free and open-source webframework and Flask- Python based micro web framework.

Django:

Django is a widely-used Python web application framework with a "batteries-included" philosophy. The principle behind batteries-included is that the common functionality for building web applications should come with the framework instead of as separate libraries.

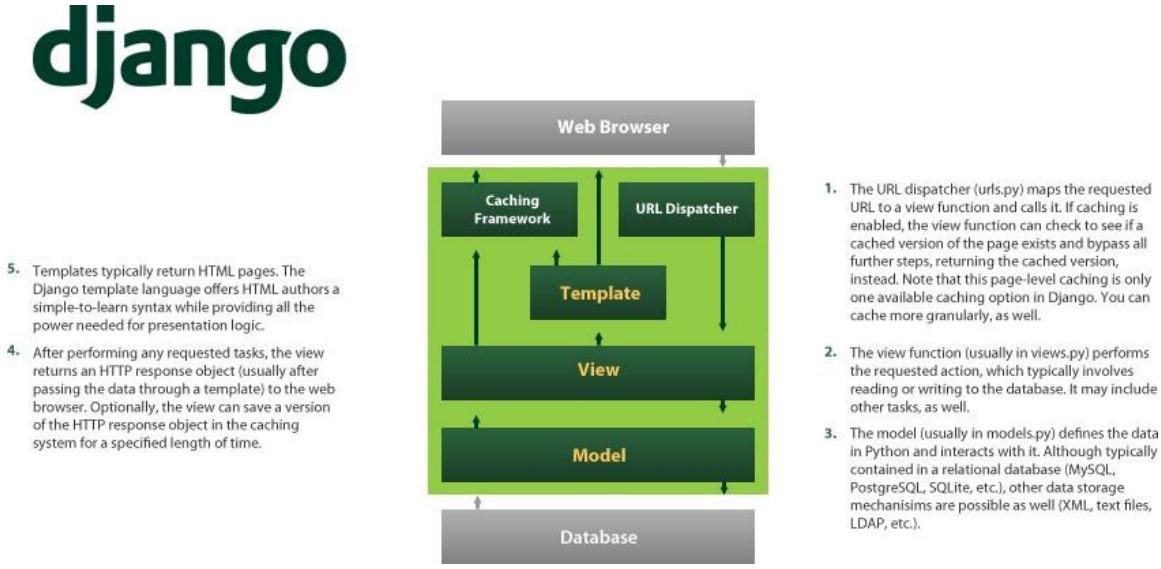
The Django project's stability, performance and community have grown tremendously over the past decade since the framework's creation. Detailed tutorials and good practices are readily available on the web and in books. The framework continues to add significant new functionality such as database migrations with each release.

Advantages of Django:

1. **Written in Python**
2. **Accelerates custom web application development**
3. **Designed as a batteries-included web framework**
4. **Supports MVC programming paradigm**
5. **Compatible with major operating systems and databases**
6. **Provides robust security features**
7. **Easy to extend and scale**
8. **Supported by a large and active community**

Disadvantages of Django:

1. **Impacts performance of small web applications**
2. **A process lacks the capability to handle multiple requests simultaneously**
3. **Makes web application components tightly-coupled**
4. **Relies heavily on the ORM system**



Flask:

Flask (source code) is a Python web framework built with a small core and easy-to-extendphilosophy. Flask is considered more Pythonic than the Django web framework because incommon situations the equivalent Flask web application is more explicit. Flask is also easy toget started with as a beginner because there is little boilerplate code for getting a simple appup and running.

Advantages:

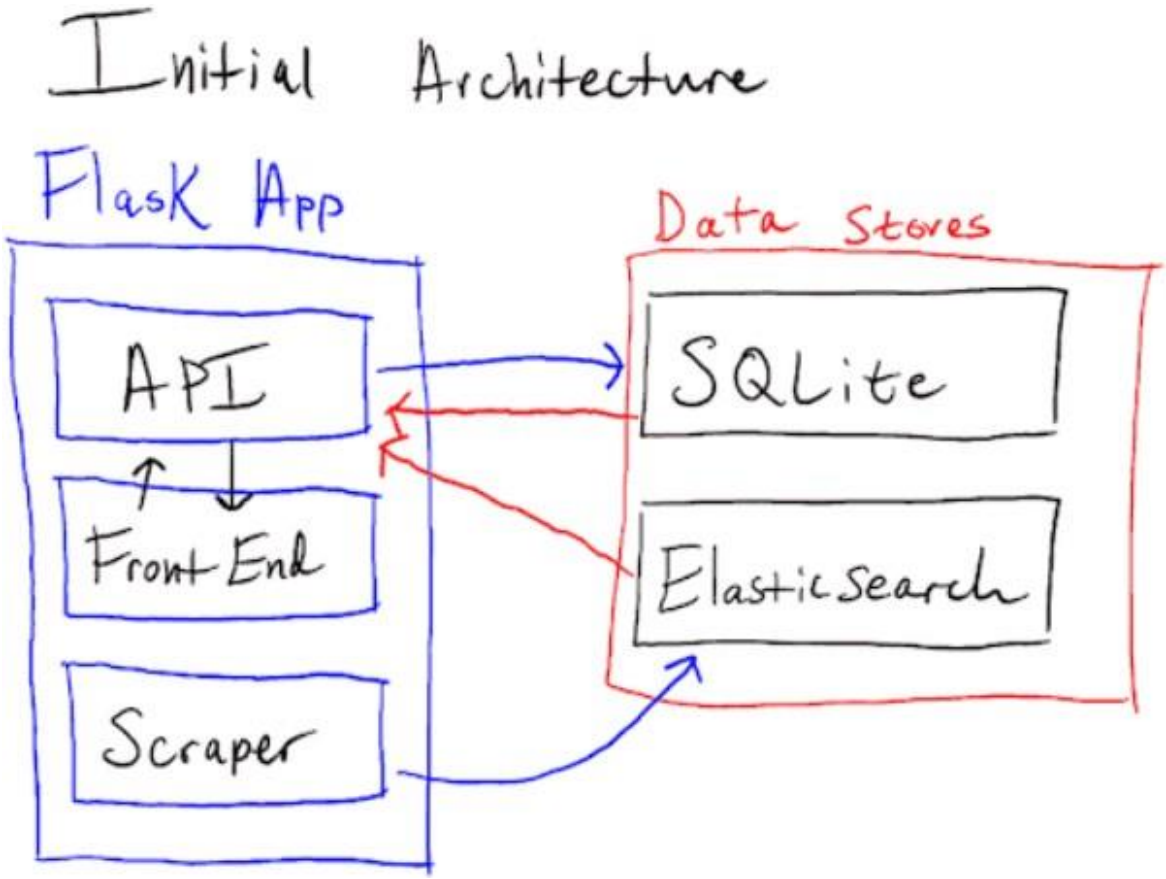
1. **Scalable: I would argue more scalable than monoliths if using modern methods. Today,applications are often running in containers or using cloud computing with auto-scaling.Applications do not typically "scale" themselves. The infrastructure scales. With a smallerapplication, it's easier to deploy instances across thousands of server easily to handleincreased traffic/load. That's part of the reason why Pinterest needed to migrate fromDjango to Flask as they grew to support more of a microservices pattern.**
2. **Simpler Development: If you understand Python well, then you'll be able to move aroundand contribute to a Flask application pretty easily. It's less opinionated so fewer standardsto learn.**
3. **Flexibility: There are very few parts of Flask that cannot be easily and safely altered becauseof its simplicity and minimality.**
4. **Performance: You can think about a micro framework being slightly more "low-level" thansomething like Django. There are fewer levels of abstraction between you and the database,the requests, the cache, etc. So performance is inherently better from the start.**
5. **Modularity: Modular code provides a huge number of benefits. With Flask, you have theability to create multiple Flask applications or servers, distributed across a large network ofservers, each with specific purposes. This creates more efficiency, better testability, andbetter performance.**

Disadvantages:

1. **Not standardized: On the flip side of being simpler, it's not very opinionated. A Pythondeveloper without Flask experience will get adjusted to a Flask application quicker than a Python developer without Django experience would get adjusted to a Django**

application.But Django is building up a large group of talent who knows the framework very well. APython developer with Django experience will get adjusted to a new Django app quickerthan a Python developer with Flask experience would get adjusted to a large Flaskapplication.

2. **Bad developers will write worse code: I like to assume that companies are not hiring "baddevelopers", but in the case that you have a younger developer training or used to havedevelopers contributing lower quality code, then it will amplify the bad code. It's easier todraw within the lines of a large, standardized framework like Django.**
3. **Fewer tools: You don't have a full toolset underneath you. So you may need to build moreon your own or search out extensions/libraries from the community.**
4. **Community: The monoliths provide such a large toolset, focused on providing solutions fora larger set of use cases out of the box, that they typically have a larger community. Thismeans more eyes on the code, more code reviews, and better-tested core code. This is alittle bit of a generalization though.**
5. **Single source of truth: One big codebase can also have its benefits from a maintenanceperspective. Developers can easily collaborate on a single repository, ensure proper codereuse, keep documentation in a single place, centralize project management/bug tracking,etc.**



In [ ]: