**1) Programming paradigms :**

Action

Agent-oriented

Array-oriented

Automata-based

Concurrent computing

Relativistic programming

Data-driven

Declarative (contrast: Imperative)

Functional

Functional logic

Purely functional

Logic

Abductive logic

Answer set

Concurrent logic

Functional logic

Inductive logic

Constraint

Constraint logic

Concurrent constraint logic

Dataflow

Flow-based

Reactive

Dynamic/scripting

Event-driven

Function-level (contrast: Value-level)

Point-free style

Concatenative

Generic

Imperative (contrast: Declarative)

Procedural

Object-oriented

Literate

Language-oriented

Natural-language programming

Discipline-specific

Domain-specific

Grammar-oriented

Intentional

Metaprogramming

Automatic

Inductive programming

Reflective

Attribute-oriented

Macro

Template

Non-structured (contrast: Structured)

Array

Nondeterministic

Parallel computing

Process-oriented

Probabilistic

Stack-based

Structured (contrast: Non-structured)

Block-structured

Object-oriented

Actor-based

Class-based

Concurrent

Prototype-based

By separation of concerns:

Aspect-oriented

Role-oriented

Subject-oriented

Recursive

Symbolic

Value-level (contrast: Function-level)

Quantum programming

https://en.wikipedia.org/wiki/Programming\_paradigm

**2) Difference between Declarative and Imperative Programming**

2.1 Declarative-də hazır olduğu üçün İmperative-ə nisbətən daha çox kod yazmağa ehtiyac yoxdu ancaq İmperative-də isə özümüz yazırıq

2.2 Declarative-də İmperative-ə nisbətən daha az kod yazdığımız üçün daha tez yazırıq sürət baxımından

2.3 Declarative-də hazir bir şəkildədir ona görə də həmin kodu başqa yerlərdə də istifadə edə bilirik ancaq İmperative-də özümüz yazdığımız üçün nisbətən daha uzun kod alınır və başqa yerlərdə istifadə etmək bir az çətin ola bilir

2.4 Declarative-də hazir bir şəkildədir ancaq İmperative-də özümüz yazdığımız üçün prosesin necə getdiyini bilməliyik və İmperative-də səhv çıxma ehtimalı özümüz yazdığımız üçün daha çoxdur

2.5 Declarative-də hazırdır ancaq məhduddur İmperative-də isə özümüz yazdığımız üçün istədiyimiz kimi başına oyun aça bilirik

https://blog.webix.com/difference-between-declarative-and-imperative-programming-with-language-examples/

**3) What are the best real life example for explaining different OOP concepts in any programming language?**

Biz burda hər şeyə bir obyekt kimi baxırıq. Hər bir obyektin də öz xüsusiyyətləri olur. Xüsusiyyətlər oxşar da ola bilər, fərqli də. Oxşar olanları bir sinifə yığırıq.

3.1 Məsələn : Bizim maşını istehsal etmək üçün bir layihə olmalıdır. Maşının da öz xüsusiyyətləri var. Bizim layihəmizdə maşının hansı xüsusiyyətləri var və onu necə ortaya qoymalıyıq o olmalıdır. Artıq layihəmiz olandan sonra maşınların sayını artıra bilərik. Bizim burdakı layihəmiz bir sinifdir. Maşın obyektdir. Maşında daşınan nədirsə o da məlumatdır.

Abstraction - Bizim üçün görməli olduğumuz Əsas xüsusiyyətləri göstərmək və qeyri-əsas xüsusiyyətləri gizlətmək.

3.2.1 Biz kiməsə zəng vurmaq istəyirik və sadəcə həmin adamın adının üstünə gəlib zəng işarəsini klik etməklə həmin adama zəng vurmuş oluruq ancaq biz arxa fonda gedən prosesləri bilmirik, biz sadəcə ekranda olan konkret şeyləri görürük yəni prosesin necə getdiyi onun gizli tərəfidir.

3.2.2 Bizə C#-da bir məsələ həll etməyi tapşırıblar. Biz də məsələni həll edirik ancaq bizim beynimizdə baş verən proseslər onlara gizlidir

Encapsulation - Kapsullaşdırmaq

3.3.1 Biz gedib aptekdən dərman (kapsula) alırıq. Həmin bu kapsula müəyyən bir maddələrin müəyyən bir nisbətdə birləşməsindən alınır. Əsində bunun özünün də içində bir Abstraction var. Hansı maddələrdən hansı nisbətdə götürülüb onu da bilmirik.

3.3.2 Bizim kod yazmağımız üçün beyin, göz, barmaq arasında bir əlaqə var bunlar birləşib bir kapsul əmələ gətirir

Encapsulation = Abstraction + Data Hiding

İnheritence - Miras alma

3.4.1 Burda xüsusiyyətlərin miras alınmasını nəzərdə tuturuq. Samsung S9 yenilənmiş versiya olsa da S2-dən qalan daşıdığı xüsusiyyətlər də var.

3.4.2 Proqramçı da bir insandır yəni insanlıqdan aldığı xüsusiyyətlər var ancaq əlavə xüsusiyyət olaraq kod yaza bilir digərlərindən fərqli olaraq.

Polymorphism - Vəziyyətə görə davranış

3.5.1 Biz müəllim, dost, ailəmiz və digər insanlarla müxtəlif cür davranırıq.

3.5.2 Biz bir proqramçıyıq və bizdən bir proqramı yazmaq, analiz və ya test etmək kimi şeylər istənilə bilər. Vəzifə, tələb və təyinata görə bunları edə bilərik.

3.5.3.1 Code Academy Digital Marketinq, Qrafik və Veb Dizayn, Proqramlaşdırma, 3D Modelləşdirmə və Vizualizasiya, Sistem Administratorluğu, MS Office kimi dərsləri verə bilir. Code Academy birdən çox vəzifəni yerinə yetirir.

3.5.3.2 Bir telefonu həm Musiqi playeri, həm GPS cihazı, həm də bir Kamera kimi işlədə bilirik

3.5.4.1 Bir insan həm müəllim, həm dost, həm də ailə üzvü ola bilir.

https://www.quora.com/What-are-the-best-real-life-example-for-explaining-different-OOP-concepts-in-any-programming-language

**4) Proqramlaşdırma paradiqmalarında hansını özünüzə daha yaxın hesab edirsiniz ?**

4.1 Xoşuma gələn 1-ci paradiqma İmperative-dir. İmperative mexanik maşın kimidi. Mexanik sürənlər üçün avtomat heç nədi ancaq avtomat sürənlər üçün mexanik çətin gəlir. Mexaniklə başlayanda maşını daha yaxşı öyrənmək olur və avtomata keçid etmək asandı. Avtomat bəzi şeyləri məhdudlaşdırır çünki özü çox şeyi edir ( Ona görə avtomat adlanır zatən ). Mexanikdə isə özümüzə uyğun etmək olur yəni özəldir. Yarış maşınları mexaniki olur. Burda da İmperative mexanik rolunu oynayır.

4.2 Xoşuma gələn 2-ci paradiqma isə Object - Oriented-dir. Obyekt yönümlü olduğundan real həyatdan nümunələr gətirmək olur. Real həyatda olan məsələləri bu paradiqma ilə həll etmək məqsədə daha uyğundur. Bir xəstəxananın, salonun, kursun, dükanın, tikinti və bir çox obyektlərin idarə etmə sistemini bu paradiqma ilə yazmaq olur.

4.3 Xoşuma gələn 3-cü paradiqma isə Functional-dır. Bu yazdığımız 3 paradiqma ən məşhur proqramlaşdırma dillərində ən çox istifadə olunan paradiqmalardı.Üstəlik Bunları bilmək digər proqramlaşdırma dillərinə keçid etməkdə işimizi asanlaşdırır.