**Object Oriented Programming-OOP (Nesne Yönelimli Programlama) :**

Gerçek dünya problermerlinin büyümesi beraberinde probleme çözüm getiren programların da büyümesine neden olmaktadır. Günümüzde yazılımların karmaşıklığının ve boyutlarının sürekli artması, yazılımda yeni kavramalrın ortaya çıkmasına neden oluyor. Nesne yönelimli programlama; gerçek yaşam probremlerinin %100 modellenebilmesi için araçlar sunmaktadır.

Klasik yapısal/fonksiyonel programlama tekniklerine yeni bir bakış acısı getiren ve farklı özellikler kazandıran Nesne Yönelimli Programlama, özellikle programların karmaşıklığını azaltmak ve güçlü çözümler sunamayı amaçlar.

Nesne yönelimli programlama; bir programlama yaklaşımıdır, disiplinidir. Bu yaklaşımda programcı, sadece programı gerçekleştiren işlevlere/görevlere odaklanmaz, bu görevi gerçekleştirmede kullanabileceği nesnelere de odaklanır. %100 nesne tabanlı bir dil olan Python'da sınıflarda (class) dahil her şey (değişkenler, liste,metot,sınıf,vb.) bir nesne olarak algılanır.

Python'da kendisine bir isim verilebilen bütün nesneler (tam sayılar,karakter dizileri,listeler, fonksiyonlar,sınıflar modüller , metotlar, vb.) eşit statüye sahiptir. Yani bütün değişkenler nesnelere atanabilir, bir liste/demet/sözlük içine yerleştirilebilir/depolanabilir,parametre olarak atanabilir.

Nesne yönelmli programlamanın temelini, class (sınıf) ve bütün class'ların örneklendirilmiş hali olan object (nesneler/objeler) oluşturur.

Python class yapısı C++ ve Modula-3 dillerinin class yapılarının bir karışımıdır. Bir Python class'ı nesne yönelimli progralamanın tüm standart ( Inheritance (Kalıtım), Encapsulation (Sarmalama), Abstract Class (Soyutlama), Polymorphism (Çok Biçimcilik)) özelliklerine doğal olarak sahiptir.

C++, Java,C#,Python,Ruby,Swift gibi günümüz modern dillerinin çoğu %100 nesne yönelimli dillerdir.

**Object ve Class'ları Anlamak :**

Class, ortak özellikleri ve davranışları olan objectlerdir, bu ortak özellik ve davranışlarını tanımlayan soyut bir kavramdır.

Python dilini bir kod fabrikası olarak düşnürsek; bu fabrikada üretilen her şeye object, bu objectlerin ansıl üretileceğini tarif eden tanımlar, kurallar/protokoller dizisine ise class denir.

Class, tasarımdan/şablondan başka bir şey değildir. Somut olarak kullanılmaz ancak bir örneği oluşturulduğunda (ki bu işleme object diyoruz) kullanılabilir.

**Class Tanımı ve Class'dan Nesne Üretimi :**

Class kendisinden objectler üretilen yapay bir modeldir/yapıdır. Bütün objectler bir classı temel alarak oluşturulurlar/türetilirler. Bir classda birden fazla object üretiebilir. Bir classdan object üretilme işlemine instantiation (örnekleme) adı verilir.

Yapılandırı (init) metot da dahil olmak üzere tüm metotlar ilk parametlere olarak objectin kendisini refere/işaret eden self işaretcisini/parametresini alırlar.

Her objectin ait olduğu bir class vardır ve bu classlar; objectlerin ortak davraışlarını ifade ederler. Objectler birbirleriyle iletişime geçebilirler, mesajlaşabilirler.

Class tanımının başlangıç ve bitişini göstermek için girintier kullanılır. Yazım kuralı olarak da class sözcüğünün tamamı küçük harflerle yazılmalıdır.

class Araba: class Araba(): class Araba(object):

pass pass pass

Python'da pass ifadesi: Java,veya C/C++ dilillerindeki gövdesiz içi boş {} fonksiyonun tanımı ile eşdeğerdir.

Class soyut bir kavram iken object somut bir kavramdır. Örneğin 'Araba' tüm arabaları tanımlayan bir üst sınıf, yani soyut bir yapıyken 'Kamyon' bu classa ait alt bir bireyi, yani somut bir objecti tanımlar. Araba; 'Renk, Kapasite,Model,Silindir' gibi özelliklere sahipse, 'Kamyon'da doğal olarak Araba classının tüm bu özelliklerine (attributes) sahip olmalıdır.

**Constructors (Yapıcı)/ Destructors (Yıkıcı) ve Özel Metotlar :**

Bir classdan bir object türertildiğinde o objecte başlangıç değeri atayabilmek için o sınıfın yapıcı fonksiyonları/metotları (constructors) tanımlanır. Yani bir classın bir örneği alındğında aynı zamanda yapıcı metodu da çalışır. Benzer şekilde türetilen objectin kontrollü bir şekilde yok edilmesi (bellekten silinmesi gerektiğinde) gerektiğinde yıkıcı fonksiyonlar/metotlar (destructors) tanımlanır.

Python, bir classdan bir örnek (object) oluşturduğunda ilk olarak otomatik yapılandırıcı yani \_ \_init\_ \_() metodu çağırılır. Bu metotdu özel olarak tanımlayarak objectlerimizi farklı değerlerle oluşturabilriz.

Java ve Python gibi diller, garbage collector (çöp toplayıcı) mekanizmasına sahiptir. Garbage Collector, işlevi sona eren objectlerin otomatik olarak bellekten siler. Bu nedenle de Python'da yıkıcı metot (destructor) kullanımına gerek yoktur. Eğer Python'da kendimiz bir objecti manuel olarak, el ile silmek/yok etmek istersek '\_ \_del\_ \_()' isimli metot çağırılır.

Özel Metotlar : Pycharm'de yazılı.

self işaretcisi : Class içinde erişilicek üyeler için kullanılır. Objectin kendisini referans etmesini sağlar. Başka bir ifade ile sözü edilen objecti doğrudan göstermek için kullanılır. Java dilidneki 'this' deyimi ile benzer işleve sahiptir.

**Class Üyeleri ve Erişim Belirteçleri :**

Bir class yapısı içerisinde yer alan her bir eleman (fonksiyonalr/metotlar, özellikler, değişkenler, veri tipleri gibi) o classın bir üyesidir. Diğer nesne yönelimli diller (Java, C#, C++,vb.) class üyelerine erişim için 'private','protected' belirteçlerini (ön eklerini) kullanırlar. Python'da ise class üyerleri benzer bir manrıkla ; açık(public), yarı gizli(semi-private) ve gizli (private) olmak üzere üçe ayrılır. Bir classın üyesinin önünde tek alt tire'\_' varsa; yarı açık (semi-private), çift alt tire '\_\_' varsa; özel (private), hiç alt tire yoksa açık (public) anlamına gelmektedir.

Python her ne kadar class üyelerini gizlemek için her ne kadar tek veya çift alt tire belirteçlerini kullanmamızı istesede bu üyeye erişmemizi tamamen engellemez fakat bizim, ilgili classı yazan kişinin kod yazım kurallarına uymasını bekler. Örneğin private bir class üyesine '\_sinifAdi\_\_gizliUye' notasyonu ile yine erişebiliriz.

**Decorators (Dekoratörler) :**

Her bir dekaratör, bir yazılım tasarım desenidir. Bu bağlamda dekoratörler; alt sınıfları doğrudan kullanmak zorunda kalmadan bir fonksiyon, metot veya sınıfın işlevselliğini/fonksyonelliğini değiştirebilmek (özellikle arttırmak) için kullanılır. Kısaca yapısını değiştirmek istemediğimiz fonksiyonların/metotların davranışlarını/işlevlerini değiştebilmek amacıyla kullanılır. Dekoratörler '@' simgesi ile tanımlanırlar.

@property Dekoratörü: property; 'özellik,nitelik' anlamında bir kelimedir. Dekoratör olarak kullanım amacı da kelime anlamına uygun olarak bir metodu özellik gibi kullanılabilecek duruma getirmektir.

@property dekoratörü; değer atama (setter), değer döndürme (getter) ve değer silme (deleter) amaçları ile kullanılabilir. İlgili özelliği @metotAdı.setter şeklinde dekore ederek ayarlarız. (Kitapda örnek kullanımı var.)

**Inheritance (Kalıtım) :**

Kalıtım, bir üst classa ait bileşenlerin alt classlara miras olarak aktarılması işlemidir.

Kalıtım ile özellikleri miras alan class, üst classın özelliklerini taşır. Bu durumda miras alınan classa üst class, miras alan classa alt class denir. Kalıtım ile bir üst classda tanımlanmış değişken veya fonksiyonlar, yeniden tanımlanmaya ihtiyaç duyulmadan bir alt classa taşınabilir.

Kalıtımla türeyen alt class, üst classa ait yeteneklere (özellik ve davraışlara) doğal olarak sahip olurken bazı yetenekleride kendine göre özelleştirebilir.

Bir alt class aynı zamanda, üst classın taşıdığı işlevleri değiştirebilir.(bu işlem 'üzerine yazma-overriding' olarak bilinir.) Tabii üzerine yazma işelminin gerçekleşebilmesi için metot isimlerinin aynı olamsına dikkat edilmeli.

Java'da bir alt classın ancak bir üst classı olabilirken Python'da birden fazla üst class olabilir ki bu duruma çoklu kalıtım (multiple inheritance) olarak adlandırlır. Bir üst classdan ise birden fazla alt class türetilebilir. (Bir superclassdan alt classlar üretme örnekleri kitapda var.)

**Multiple Inheritance :**

Python'da bir alt classın birden fazla üst classı olabilir bu durum çoklu kalıtım olarak bilinir. Çoklu kalıtım birden fazla üst classa ait özelliklerin miras alan alt classa aktarılması işlemidir.

**Encapsulation and Data Hiding (Sarmalama ve Veri Gizleme) :**

Kapsülleme veya paketleme olarakta isimlendirilen sarmalama (encapsulation); bir classın içerğinin gizlenerek dışarıya sadece istenilen özellik veya fonskiyonların gösterilmesi işlemidir. Sarmalama (encapsulation) ile veri gizlenir, yani veriye doğrudan erişilemez, yalnızca getter/setter metotlar/fonksiyonlar ile erişilebilir ve sınırlı değişiklikler yapılmasına izin verilir. Her object, belli verileri tutar ve belli işlevler görür. Bir class, aslında başka classların kullanılması için çeşitli özellik ve fonksiyonlar barındıran bir yapıdır. Ancak bir classdaki bütün özellik ve fonksiyonalrın dışarıdan bilinmesi veya kullanıması gerekmez. Hatta bazı özellik ve fonksiyonlara erişim, classın sağlamlığı/güvenliği için tehlikeli olabilir. Sarmalama ile bir sınıf kendi iç bütülüğünü gizleyebilir ve koruyabilir. Bir classın dışarıdan sadece gereken özellik ve fonksiyonlarıyla götürülmesi ayrıca basitlikte sağlamaktadır. Sarmala ile bir sınıfa ait detaylar sarmalanıp gizlenerek classda etkileşim, kontrollü bir şekilde sağlanır. Örnek olarak; bir aracın bazı parçaları (motor,şanzıman) kaputun içinde paketlenerek, gizlenerek korunur.

**Getter/Setter (get/set Ön Ekli) Fonksiyonlar :**

Sarmalama işlemi için genellikle getter ve setter (get ve set ön eki ile başlayan) fonksiyonlar/metotlar kullanılır. set ön ekli fonksiyonlar ; parametre alan ve aldığı parametre ile private üyenin değerini set eden değiştiren fonksiyonlardır. get ön ekli fonksiyonlar ise return fonksiyonu ile private üyenin değerini geri dödüren fonksiyondur.

**Polymorphism (Çok Biçimlilik) :**

Yunanca 'poly' (çok) ve 'morpho''s' (şekil) kelimelerinin birleişimden oluşan Polymorphism kelimesi çok biçimlilik veya çoklu işlev anlamına gelmektedir. Çok biçimlilik nesnelerin aynı olaylara farklı tepkiler verebilme yeteneği, başka bir tanıma göre ise, bir objectin davranış şekillerinin duruma göre değiştirebilme yeteneğidir.

Çok biçimlilik programlarda daha basit bir görünüm (bir çok davranış biçimi için tek bir arayüz) sağlar buda programın test ve hata kontrolünü kolaylaştırır.

Çok biçimlilik, aralarında kaıtım ilişkisi bulunan üst ve alt classlarda tanımlanan ortak işlevler üzerine inşa edilir. Bu yapıda ortak ffonksiyonun üst ve alt classlarda içeriği farklı tanımlanır.

Her iki nesnede de aynı isimdeki metodu kullanıp farklı çıktılar elde etme işlemine **Polymorphism** yani**Çok Biçimlilik** denilmektedir. Bizler, aynı isme sahip birden fazla metot oluşturabilir fakat her metodun işlevini birbirinden farklı şekilde düzenleyebiliriz.

**Abstract Class (Soyut Sınıf) :**

Soyutlama kullanıcan gereksiz bilgileri gizleme ve sadece gerekli özellikleri gösterme işlemidir. Başka bir ifade ile gerçek yaşam probleminin gereksiz detaylardan ayıklanarak bilgisayar ortamına modellenmesidir. soyutlama ve sarmalama kavramları karıştırılsa da sarmalama daha çok nesnenin kullanım süreci ile ilgilenirken, soyutlama tasarım süreci ile ilgilenir.

Abstract Class çok biçimli class yapısında, kendisinde nesne türetilmeyen, sadece kalıtım amaçlı kullanılan bir üst classdır. Soyut sınıf içerisinde bir veya daha fazla metot içeren sınıftır.

Bazen en üst classın kendisinden türeyecek diğer alt classlar için ortak özelliklere, davranışlara uyacak somut özellik veya fonksiyonlarını tanımlamak başlangıçta mümkün olmayabilir. Çok biçimlilik kavramı açıklanırken verilen 'Hayvan' sınıfının ortak işlevi/fonksiyonları 'konusma()', 'beslenme()' olabilir. 'köpek', 'kedi' onun alt sınıfları olmalarına rağmen her bir hayvanın 'Konusma()', 'beslenme()' gibi metotları/fonksiyonları farklıdır. Bu seple hayvan üst classında konusma beslenme gibi fonksiyonalrın gövdesini tanımlamadan bırakırız (pass). Böylece hayvan sınıfı soyut bir sınıf olmuş olur.

Soyut bir classın, alt classları türedikleri soyut classın tüm fonksiyonlarını ya gerçekleştirmek ya da üzerine yazmak (ovveride etmek) zorundadır.

Python'da soyut temel classları (abstract base classes) tanımlamak için abc modülü kullanılır. Soyut metot tanımı için @abstractmethod dekoratörü kullanılır.

**Super Deyimi :**

Alt (sub) sınıftaki her nesneden üst(süper) sınıftaki nesneye erişmek için kullanılır. Özellikle kullanım amacı; miras alınan üst sınıf yapılandırıcısına çağrıda (super(),\_\_init\_\_() gibi) bulumaktır. Böylece üst sınıf özellik ve metotları üzerinde değişklik yaparken, mevcut alt sınıfın özelliklerini de korumuş oluruz.