2024-2025 EĞİTİM VE ÖĞRETİM YILI

PROGRAMLAMA DİLLERİ MODÜLÜ (PYTHON)

1. ÜNİTE

PYTHON PROGRAMLAMAYA GİRİŞ

1.1. Python programlama diliyle çalışılacak ortamlar

IDE (Integrated Development Environment) nedir?

IDE "Tümleşik Geliştirme Ortamı" demektir.

IDE yazılım geliştirme aşamasında geliştiriciye birçok kullanışlı araç sunarak daha kolay ve etkili şekilde yazılım geliştirmesine yardımcı olan yazılımlardır.

Python IDE türleri:

Python için birçok editörü kullanabiliriz. Bu tamamen kullanım alışkanlıkları ve kullanım kolaylığı gibi faktörlere bağlıdır. Bu editörlerden bazıları şunlardır.

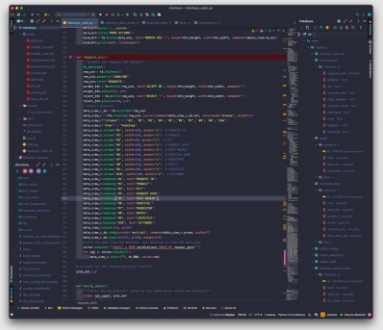
- Python IDLE
- Pycharm
- PyScripter
- PyDev
- Eric Python
- Jupyter Notebook
- Atom
- Spyder
- Sublime text
- Visual Studio Code vb...

Python IDLE

- Python.org web sitesinde yer alan ücretsiz bir Python editörüdür. Başlangıç seviyesi için en temel editördür.
- Küçük kodları IDLE ekranına yazabilirsiniz ancak daha geniş kapsamlı kodlar için sol üst köşede File [Dosya] menüsüne ait olan New File [Yeni Pencere] seçeneğine tıklamanız gerekir.

Python IDLE

- Burada beyaz bir ekranla karşılaşacaksınız. İşte asıl kodları buraya yazmanız gerekecektir.
- Yazdığınız kodları kaydetmek içinse File [Dosya] menüsüne ait olan Save As [Farklı Kaydet] seçeneğine tıklamanız gerekir.
- Kodları direk çalıştırmak için Run [Çalıştır] menüsüne ait olan Run Module seçeneğine ya da klavyeden direk F5 tuşuna tıklayabilirsiniz.



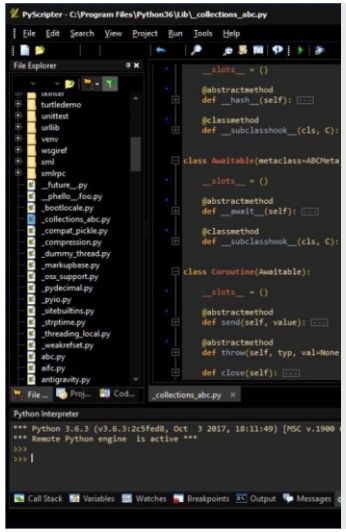
PyCharm

Avantajlar:

- Zengin özellikler (otomatik tamamlama, hata ayıklama, versiyon kontrol entegrasyonu)
- Profesyonel ve güçlü bir IDE
- Çapraz platform (Windows, macOS, Linux)

Dezavantajlar:

- Ağır olabilir, özellikle daha düşük sistem kaynaklarında
- Ücretli bir profesyonel sürümü var (Community sürümü ücretsiz olsa da bazı özellikler eksik)



PyScripter

Avantajlar:

- Hafif ve hızlı
- Ücretsiz ve açık kaynak
- Hata ayıklama, kod profilleme gibi kullanışlı araçlar

Dezavantajlar:

- Sadece Windows'ta çalışıyor
- Daha az özellik ve eklenti desteği

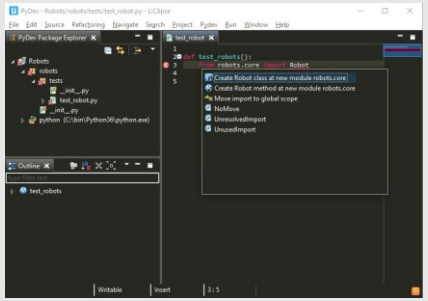
PyDev (Eclipse ile birlikte)

Avantajlar:

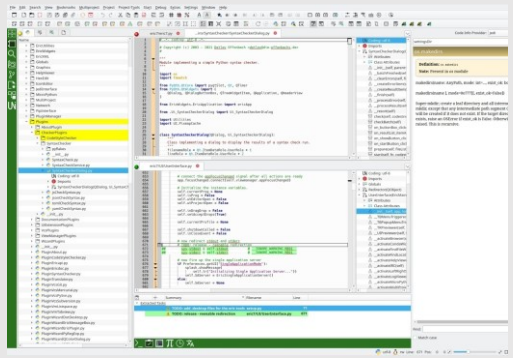
- Eclipse, yazılım geliştiricilerin diğer programlama dillerinde yazılmış kodları geliştirmelerine ve test etmelerine olanak sağlayan eklentileriyle bilinen ücretsiz, Java tabanlı bir geliştirme platformudur.
- Eclipse ile entegre, çoklu dil desteği (Java, C++, vb.)
- Gelişmiş hata ayıklama ve otomatik tamamlama özellikleri
- Ücretsiz

Dezavantajlar:

- Eclipse'in karmaşık yapısı nedeniyle başlangıç için öğrenmesi zor olabilir
- Ağır ve yavaş olabilir



Eric Python



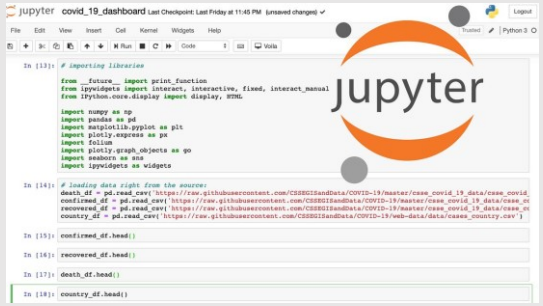
Avantajlar:

- Zengin özellikler (hata ayıklama, versiyon kontrolü, proje yönetimi)
- Hafif ve hızlı

Dezavantajlar:

- Arayüzü daha az modern ve kullanışsız olabilir
- Büyük topluluğa sahip değil, destek az

Jupyter Notebook



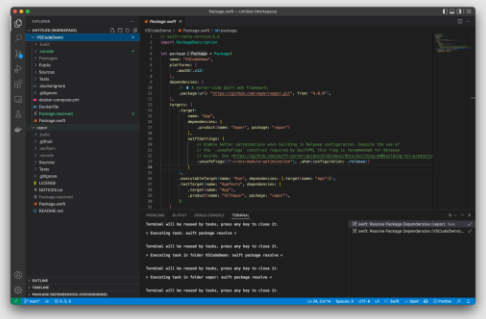
Avantajlar:

- Veri bilimi ve makine öğrenmesi için mükemmel (etkileşimli not defteri, görselleştirme desteği)
- Web tabanlı ve kod parçacıklarıyla çalışmak kolay
- Markdown desteği, not tutma ve rapor oluşturma için uygun

Dezavantajlar:

- Tam bir IDE değil, büyük projeler için uygun değil
- Hata ayıklama ve kod tamamlama gibi bazı özellikler sınırlı

Visual Studio Code (VS Code)



Avantajlar:

- Hafif, hızlı ve özelleştirilebilir
- Çok sayıda uzantı ve eklenti desteği Çapraz platform, ücretsiz

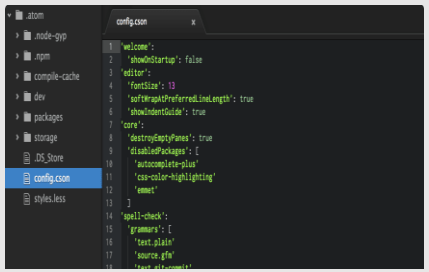
Dezavantajlar:

- Tam bir IDE'den ziyade bir metin editörü olarak kabul edilir (özellikleri genişletmek için uzantılara ihtiyaç duyulur)
- Büyük projelerde performans sorunları yaşanabilir

Atom

Avantajlar:

- Atom tamamen ücretsizdir ve açık kaynak kodludur.
- GitHub Entegrasyonu:
- **Windows**, **macOS** ve **Linux** üzerinde çalışır.
- **Teletype**: Farklı bilgisayarlardaki kullanıcıların aynı proje üzerinde eş zamanlı olarak çalışmasına olanak tanır.



Dezavantajlar:

- Tam bir IDE'den ziyade bir metin editörü olarak kabul edilir (özellikleri genişletmek için uzantılara ihtiyaç duyulur)
- Büyük projelerde performans sorunları yaşanabilir

Sublime Text

Avantajları:

1. **Hız ve Performans**
2. **Düşük Bellek Kullanımı**
3. **Çoklu Platform Desteği**
4. **Zengin Eklenti Desteği**: **Package Control** ile birçok eklentiye ve özelleştirme seçeneğine sahiptir. Eklenti kurulumları hızlı ve basittir.
5. **İmleç Desteği**: Aynı anda birden fazla imleç kullanarak birden fazla konumda düzenleme yapabilirsiniz.

- **Dezavantajları:**

6. **Ücretli**: Sublime Text ücretsiz deneme sürümü sunsa da, tam sürümünü kullanmak için bir **lisans** satın almak gerekiyor.
7. **IDE Özellikleri Zayıf**: Sublime Text, tam anlamıyla bir **IDE** değildir. Derleme, hata ayıklama ve test gibi gelişmiş özellikler için eklentilere bağımlıdır.
8. **Sınırlı Özelleştirme**: Özelleştirme seçenekleri geniş olsa da, Atom kadar derin ve kapsamlı bir paket sistemi sunmaz.

Sublime Text

Avantajları:

Hız ve Performans

Düşük Bellek Kullanımı

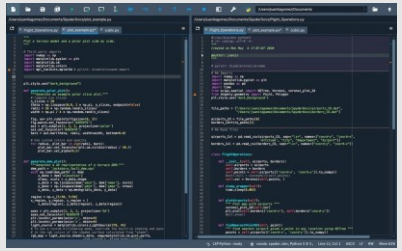
Çoklu Platform Desteği

Zengin Eklenti Desteği: **Package Control** ile birçok eklentiye ve özelleştirme seçeneğine sahiptir.

İmleç Desteği: Aynı anda birden fazla imleç kullanarak birden fazla konumda düzenleme yapabilirsiniz.

Dezavantajları:

1. **Ücretli**: Sublime Text ücretsiz deneme sürümü sunsa da, tam sürümünü kullanmak için bir **lisans** satın almak gerekiyor.
2. **IDE Özellikleri Zayıf**: Sublime Text, tam anlamıyla bir **IDE** değildir. Derleme, hata ayıklama ve test gibi gelişmiş özellikler için eklentilere bağımlıdır.
3. **Sınırlı Özelleştirme**: Özelleştirme seçenekleri geniş olsa da, Atom kadar derin ve kapsamlı bir paket sistemi sunmaz.



Spyder

Avantajları:

1. **Python'a Odaklı**: Python geliştiricileri için optimize edilmiştir; bilimsel hesaplamalar ve veri analizi için popülerdir.
2. **Entegre Araçlar**: İstatistik ve veri analizi için entegre araçlar, grafik gösterimi ve veri keşfi için güçlü özellikler sunar.
3. **Değişken İnceleme**: Değişkenleri görsel olarak inceleyebilir ve yönetebilirsiniz, veri setleriyle çalışmak için idealdir.
4. **Hata Ayıklama**: Güçlü hata ayıklama araçları ile kod üzerinde detaylı kontrol sağlar.

Dezavantajları:

1. **Performans Sorunları**: Büyük projelerde ve karmaşık veri işlemlerinde yavaşlayabilir.
2. **Python'a Özel**: Yalnızca Python dili ile sınırlıdır, başka dillerde çalışmak isteyenler için esnek değildir.
3. **Daha Az Özelleştirilebilirlik**: Diğer metin editörleri ve IDE'lere kıyasla daha sınırlı eklenti ve tema desteği sunar.
4. **Ağır Arayüz**: Kullanıcı arayüzü daha karmaşık ve ağırdır, bu da bazı kullanıcılar için kafa karıştırıcı olabilir.

Python IDE türleri

Ayrıca online IDE'lere de kod yazıp çalışab**ilirsiniz. Yapmanız gereken hangi programlama diliyle çalışmak istediğinizi seçip sonrasında özgürce kod yazmaktır.**

Örnek online IDE'ler

- ideone.com
- codechef.com
- compileonline.com
- tutorialspoint.com

Python seçeneği seçildi

Derlenen (Compiled) ve Yorumlanan (Interpreted) Diller:

Bir programlama dilinin derlenen veya yorumlanan dil olması, programın nasıl çalıştırıldığıyla ilgilidir.

Bu iki tür dilin farkları, çalışma performansı, taşınabilirlik ve hata ayıklama süreçlerinde kendini gösterir.

Derlenen Diller

- Derlenen dillerde, yazılan kod önce bir derleyici (compiler) tarafından makine diline çevrilir.
- Bu işlem sonucunda, bağımsız bir **çalıştırılabilir dosya** (örneğin, .exe dosyası) elde edilir. Bu dosya, hedef makinede direkt olarak çalıştırılabilir.
- Derleyici, tüm kaynak kodu tek seferde alır ve makine diline çevirir.

Derlenen dillerin özellikleri:

1. **Yüksek Performans:** Derlenmiş kod, doğrudan makine dilinde çalıştığı için daha hızlıdır.
2. **Hata Tespiti:** Derleme sırasında hatalar tespit edilir ve program çalışmadan önce düzeltilir.
3. **Tekrar Derleme İhtiyacı:** Kaynak kodda yapılan bir değişiklik sonrasında kodun tekrar derlenmesi gerekir.

Derlenen dillerin;

Avantajları:

- Yüksek performans
- Çalıştırılabilir dosyalar her yerde çalıştırılabilir (aynı işletim sisteminde)

Dezavantajları:

- Derleme süreci zaman alabilir.
- Farklı platformlarda çalıştırmak için kodu her platforma özel derlemek gerekir.

Derlenen diller;

- **C**
- **C++**
- **Rust**
- **Go**

Yorumlanan Diller

- Yorumlanan dillerde ise kaynak kod, her çalıştırıldığında bir yorumlayıcı (interpreter) tarafından satır satır işlenir.
- Kod, her çalıştırıldığında tekrar yorumlanır ve anında çalıştırılır.
- Yorumlanan diller genellikle daha esnektir, ancak performans açısından daha yavaş olabilirler.

Yorumlanan Diller

Özellikler:

Anında Çalıştırma: Kaynak kod, derlenmeden doğrudan yorumlayıcı tarafından çalıştırılır.

Taşınabilirlik: Yorumlanan diller genellikle platformdan bağımsızdır. Aynı kod, farklı işletim sistemlerinde yorumlayıcı ile çalıştırılabilir.

Daha Yavaş Çalışma: Her seferinde yorumlanması gerektiği için derlenen dillere göre daha yavaş olabilir.

Yorumlanan Diller

Avantajlar:

- Daha hızlı geliştirme ve hata ayıklama
- Farklı platformlarda çalıştırılabilir, yeniden derleme gerekmez

Dezavantajlar:

- Yavaş çalışma performansı
- Her çalıştırmada yorumlayıcıya ihtiyaç duyulması

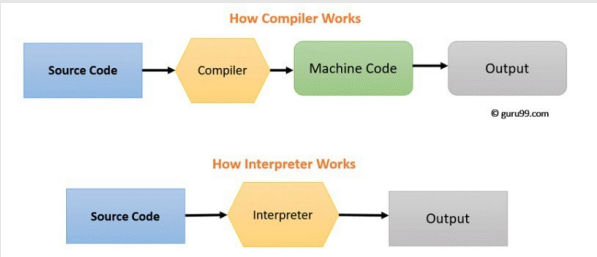
Yorumlanan Diller

- Python
- JavaScript
- Ruby
- PHP

Derlenen ve Yorumlanan Diller Arasındaki Farklar:

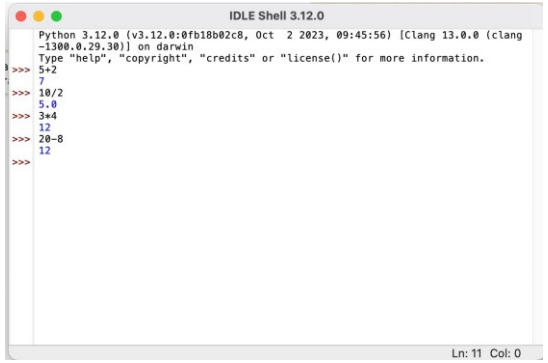
1. **Çalıştırma Şekli**: Derlenen dillerde kod önce makine diline çevrilir ve bu şekilde çalıştırılır. Yorumlanan dillerde ise kod doğrudan satır satır çalıştırılır.
2. **Hız**: Derlenen diller genellikle daha hızlı çalışır çünkü doğrudan makine dilinde çalıştırılırlar. Yorumlanan diller ise her satırı yorumladıkları için daha yavaş olabilir.
3. **Hata Ayıklama**: Yorumlanan dillerde hatalar çalışma anında daha kolay fark edilebilir. Derlenen dillerde ise hatalar genellikle derleme aşamasında tespit edilir.

Derlenen ve Yorumlanan Diller Arasındaki Farklar:



1.2. Farklı entegre geliştirme ortamlarını analiz ederek kendisine en uygun çalışma ortamını seçer.

Python IDE'yi açınız ve aşağıdaki komutları yazarak enter tuşuna basınız.



Kaynak: onl

Yorum Satırları

Python'da **yorum satırları**, kodun çalışmasını etkilemeden açıklamalar eklemek için kullanılır.

Yorumlar, kodun anlaşılabilirliğini artırmak ve gelecekteki kullanıcılar (veya kendiniz) için rehberlik sağlamak amacıyla yazılır.

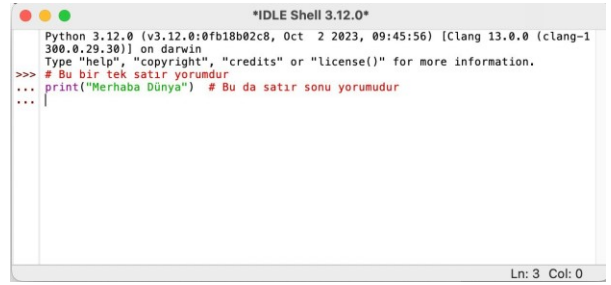
Kaynak: onu

Tek Satırlık Yorum

Python'da tek satırlık bir yorum oluşturmak için **#** (diyez) işareti kullanılır.

Bu işaretin sağındaki metin yorum olarak kabul edilir ve Python bu kısmı çalıştırmaz.

ÖRNEK



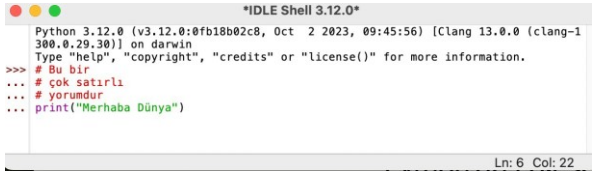
Kaynak: onu

Çok Satırlık Yorum

Python'da çok satırlı yorumlar için özel bir sembol yoktur, ancak birkaç yol vardır.

En yaygın yöntemlerden biri, her satırın başına **#** koyarak birkaç satırı yorum haline getirmektir.

ÖRNEK



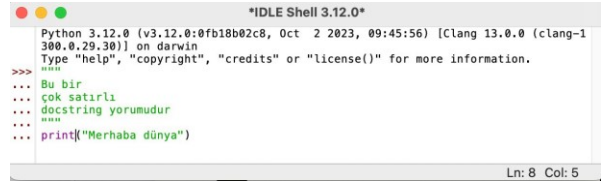
Kaynak: onu

Çok Satırlık Yorum

Alternatif olarak, **docstring** (üçlü tırnak """ """ veya ''' ''') kullanılarak çok satırlı yorum yapılabilir, ancak bu teknik aslında yorum yerine **belge dizgisi** (docstring) olarak kabul edilir.

Docstring, genellikle fonksiyonlar veya sınıflar için açıklama sağlamak amacıyla kullanılır. Yorum amacıyla kullanılabilir ama Python tarafından derlenir, bu yüzden sadece açıklama amacıyla docstring kullanmak önerilmez.

ÖRNEK



Kaynak: onu

print() fonksiyonu

Python'da **print()** fonksiyonu, bir programın çıktısını ekrana yazdırmak için kullanılır ve programlamada temel bir yapı taşıdır.

print() fonksiyonu

print() fonksiyonunun önemi ve neden gerekli olduğuna dair bazı noktalar:

Anlık Geri Bildirim Sağlar

print(), bir programın çalışırken neler yaptığını ve hangi sonuçları ürettiğini görmenin en basit yoludur. Programın doğru çalışıp çalışmadığını kontrol etmek için çıktıları ekranda görüntülemek hayati önem taşır. Bu, özellikle hata ayıklama (debugging) sırasında kodun belirli bir noktada nasıl davrandığını görmek için kullanılır.

Kaynak: onl

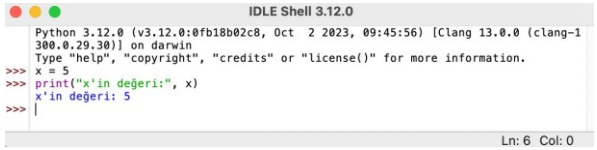
print() fonksiyonu

print() fonksiyonunun önemi ve neden gerekli olduğuna dair bazı noktalar:

Değişkenlerin Durumunu İzler

Bir program çalıştığında, değişkenlerin değerlerinin zamanla nasıl değiştiğini görmek gerekebilir. **print()**, değişkenlerin değerlerini ekrana yazarak programın adım adım ne yaptığını anlamaya yardımcı olur. Bu, kodun izlenebilirliğini artırır.

Örnek:



Kaynak: onl

print() fonksiyonu

print() fonksiyonunun önemi ve neden gerekli olduğuna dair bazı noktalar:

Hata Ayıklama (Debugging)

Programda hata olduğunda, **print()** komutuyla kodun belirli bölümlerini izleyerek hatanın nerede meydana geldiğini kolayca bulabilirsiniz.

Kaynak: onl

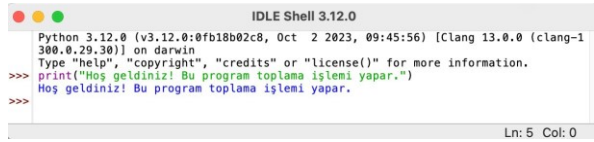
print() fonksiyonu

print() fonksiyonunun önemi ve neden gerekli olduğuna dair bazı noktalar:

Kullanıcıyla Etkileşim

print(), programın çıktısını kullanıcıya sunarak bir tür etkileşim sağlar. Kullanıcıya bilgilendirici mesajlar, sonuçlar veya yönergeler verebilirsiniz.

Örnek:



Kaynak: onl

print() fonksiyonu

print() fonksiyonunun önemi ve neden gerekli olduğuna dair bazı noktalar:

Kodun Anlaşılabilirliğini Artırır

Kodun çalışması sırasında çıktıları görmek, hem geliştirici hem de başka biri tarafından anlaşılmasını kolaylaştırır. Yazdırma işlemleri, kodun akışını ve mantığını daha net hale getirir.

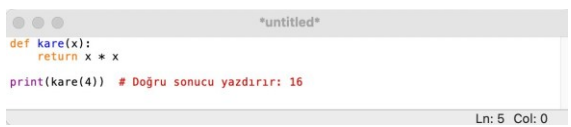
Kaynak: onl

print() fonksiyonu

print() fonksiyonunun önemi ve neden gerekli olduğuna dair bazı noktalar:

Test ve Doğrulama Aracı

print(), küçük testler yaparak bir programın düzgün çalışıp çalışmadığını kontrol etmenin basit ve hızlı bir yoludur. Bu, özellikle bir fonksiyonun doğru sonuçlar üretip üretmediğini kontrol etmek için gereklidir.



Kaynak: onl

input() fonksiyonu

Python'da, kullanıcıdan veri almak için **input()** fonksiyonu kullanılır.

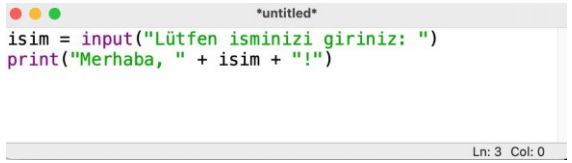
Bu fonksiyon, kullanıcıdan bir veri girişi bekler ve kullanıcı veriyi girdikten sonra bu veriyi bir **string** (metin) olarak döner.

Kaynak: onl

input() fonksiyonu

Basit Bir Kullanıcı Girişi

Kullanıcıdan isim almak ve ekrana yazdırmak:



Çıktı:
Lütfen isminizi giriniz: Onur
Merhaba, Onur!

Kaynak: onu

input() fonksiyonu

Kullanıcıdan Sayısal Veri Alma

input() fonksiyonu varsayılan olarak string döner, bu yüzden sayısal veri almak için dönüştürme işlemi yapılmalıdır:

```
yas = int(input("Kaç yaşındasınız? "))
print(f"10 yıl sonra {yas + 10} yaşında olacaksınız.")
```

Çıktı:
Kaç yaşındasınız? 15
10 yıl sonra 25 yaşında olacaksınız.

Kaynak: onu

input() fonksiyonu

Kullanıcıdan Birden Fazla Veri Alma

Kullanıcıdan aynı anda birden fazla değer alıp bunları değişkenlere atayabilirsiniz:

```
ad = input("Adınızı giriniz: ")
yas = int(input("Yaşınızı giriniz: "))
print(f"{ad}, {yas} yaşındasınız.")
```

Çıktı:
Adınızı giriniz: Duru
Yaşınızı giriniz: 17
Duru, 17 yaşındasınız.

Kaynak: onu

input() fonksiyonu

Matematiksel İşlem Yapma

Kullanıcıdan alınan sayıları matematiksel işlemler için kullanabilirsiniz:

```
sayi1 = float(input("Birinci sayıyı giriniz: "))
sayi2 = float(input("İkinci sayıyı giriniz: "))
toplam = sayi1 + sayi2
print("Girilen sayıların toplamı", toplam)
```

Çıktı:
Birinci sayıyı giriniz: 20
İkinci sayıyı giriniz: 30
Girilen sayıların toplamı: 50.0

Kaynak: onu

input() fonksiyonu

Koşullu Yapılarla Kullanıcı Girdisi

Kullanıcıdan alınan verilere göre koşullar belirleyebilirsiniz:

```
sifre = input("Şifrenizi giriniz: ")
if sifre == "1234":
    print("Şifre doğru, giriş yapıldı.")
else:
    print("Hatalı şifre!")
```

Çıktı:
Şifrenizi giriniz: 1234
Şifre doğru, giriş yapıldı.

Çıktı:
Şifrenizi giriniz: 1235
Hatalı şifre!

Kaynak: onl

1.3. Değişkenlerde tutulacak veriler

Python'da **veri türleri** (data types), bir değişkenin hangi türde veri saklayacağını belirten kategorilerdir.

Programlama sırasında farklı türlerde verilere ihtiyaç duyulur ve bu türlerin doğru yönetilmesi, hem bellek yönetimi açısından hem de işlemler açısından önemlidir.

Veri türü ne demek?

Veri türü, bir değişkenin saklayabileceği verilerin biçimini ve bu verilerle hangi işlemlerin yapılabileceğini tanımlar.

Python, dinamik tür yapısına sahiptir, yani bir değişkenin türü tanımlandıktan sonra otomatik olarak belirlenir. Örneğin, sayısal işlemler için bir integer (tamsayı), metinsel işlemler için string (metin) kullanılır.

Hangi veri türünün kullanılacağını bilmek, programın performansını ve doğruluğunu doğrudan etkiler.

Veri Türlerinin Çeşitli Olmasının Nedenleri:

Farklı Veri Tiplerini Temsil Etme İhtiyacı:

Farklı türdeki bilgilerin
(sayılar, metinler, mantıksal değerler, listeler) programlarda temsil edilmesi gerekir.

Bellek Yönetimi:

Her veri tipi bellekte farklı miktarda yer kaplar ve farklı işlemler gerektirir. Örneğin, bir tamsayı daha az yer kaplarken, bir liste daha fazla yer kaplar.

Doğru İşlemler İçin:

Her veri türü kendine özgü işlemlerle çalışır.

Örneğin, sayılarla toplama yapabilirken, metinlerle birleştirme işlemi yapabiliriz.

Python Veri Türlerinin İsimleri:

Sayısal Veri Türleri (Numeric Types):

int: Tamsayıları temsil eder.
Örnek: x = 10

float: Ondalıklı sayıları temsil eder.
Örnek: y = 10.5
z = 2.0

complex: Karmaşık sayıları temsil eder.

Örnek: z = 3 + 5j

Python Veri Türlerinin İsimleri:

Metinsel Veri Türü (Text Type): str:
Metinleri temsil eder (string).

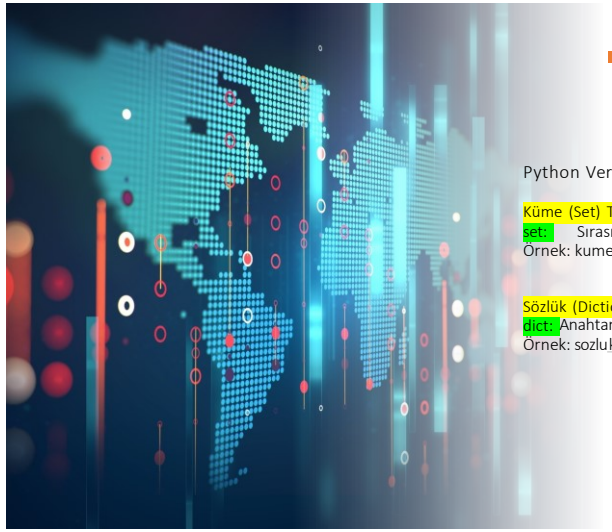
Örnek: isim = "Python "

Mantıksal Veri Türü (Boolean Type):
bool: Doğru veya yanlış değerleri saklar (True ya da False).
Örnek: aktif = True

Dizi (Sequence) Türleri:
list: Sıralı, değiştirilebilir veri koleksiyonu.
Örnek: liste = [1, 2, 3, "elma"]

tuple: Sıralı, değiştirilemez veri koleksiyonu.
Örnek: demet = (1, 2, 3)

range: Belirli bir aralıktaki sayıları temsil eder.
Örnek: aralik = range(5)



Python Veri Türlerinin İsimleri:

Küme (Set) Türleri:
set: Sırasız, benzersiz veri koleksiyonu.
Örnek: kume = {1, 2, 3}

Sözlük (Dictionary) Türü:
dict: Anahtar-değer çiftlerinden oluşan koleksiyon.
Örnek: sozluk = {"isim": "Ali", "yas": 25}

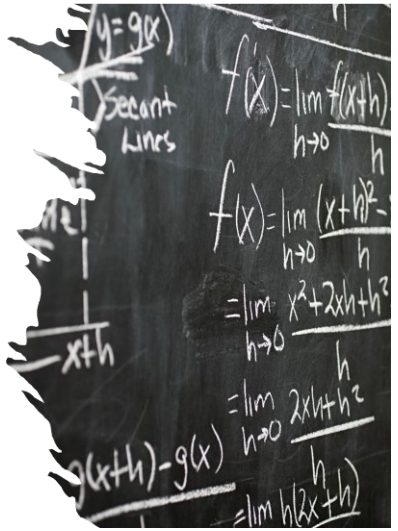
Değişken Tanımlama Kuralları:

İsimlendirme:

Bir değişken adı, harf (a-z, A-Z) veya alt çizgi (_) ile başlamalıdır.

Doğru: isim, _degisken, yas_23

Yanlış: 23yas, -ad



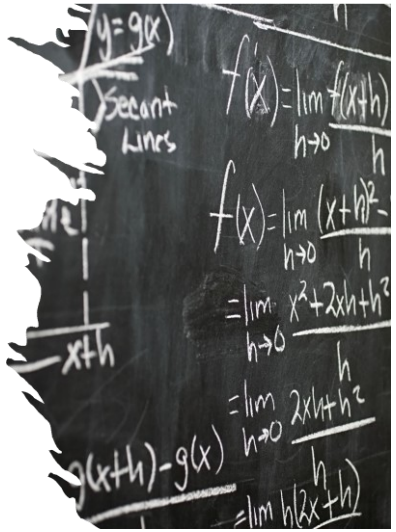
Değişken Tanımlama Kuralları:

İçeriği:

Değişken adı, harfler, sayılar (0-9), ve alt çizgi (_) içerebilir.

Doğru: degisken1, yas_23

Yanlış: isim!, ad soyad (boşluk ve özel karakterler içeremez)



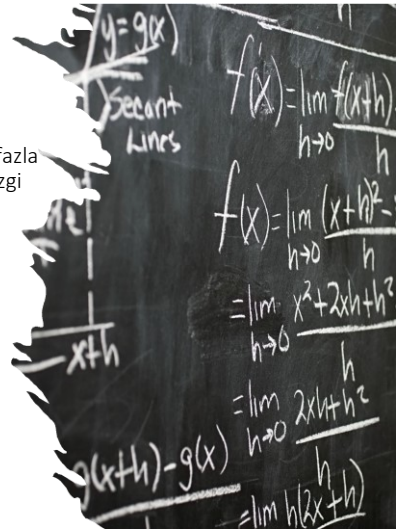
Değişken Tanımlama Kuralları:

Boşluklar Yok:

Değişken adlarında boşluk kullanılamaz. Eğer birden fazla kelimeden oluşan bir değişken ismi gerekiyorsa, alt çizgi (_) kullanılabilir veya kelimelerin her biri büyük harfle yazılabilir (CamelCase).

Doğru: ogrenci_adi, OgrenciAdi

Yanlış: ogrenci adi

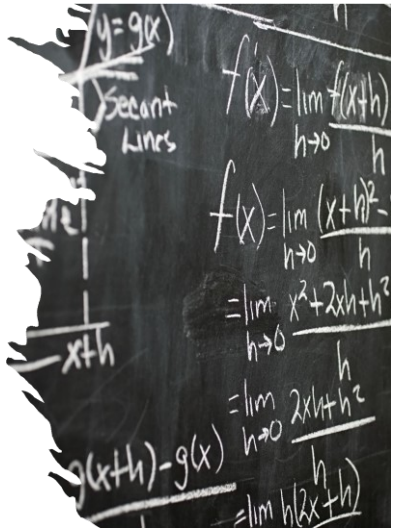


Değişken Tanımlama Kuralları:

Python Anahtar Kelimeleri Kullanılamaz:

Python'un kendi komutları olan anahtar kelimeler değişken adı olarak kullanılamaz. (örneğin: if, for, while, True, None).

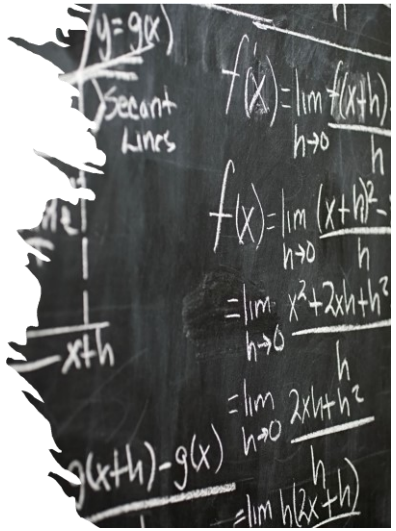
Yanlış: if = 10, True = 5



Değişken Tanımlama Kuralları:

Büyük-Küçük Harf Duyarlılığı:

Python'da değişkenler büyük/küçük harf duyarlıdır. Yani, degisken ve Degisken farklı değişkenler olarak kabul edilir.



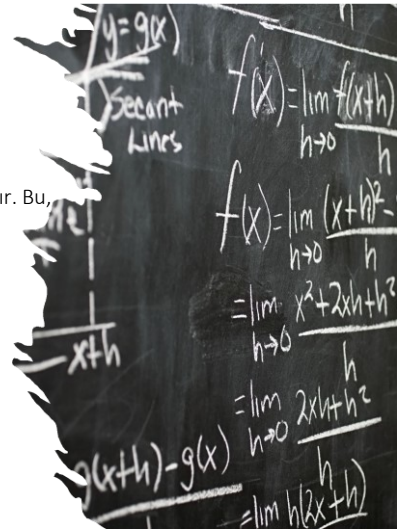
Değişken Tanımlama Kuralları:

Açıklayıcı İsimler Kullanma:

Değişken adları, sakladıkları veriyi açıklayıcı olmalıdır. Bu, kodun okunabilirliğini artırır.

Doğru: ogrenci_sayisi, ortalama_not

Yanlış: x, abc



Python için Ayrılmış Kelimeler			Python için Doğru Belirteçler	Python için Yanlış Belirteçler
and	del	from		☒ ara-toplam (- sembolü kullanılamaz)
as	elif	global	☑ x	☒ ilk değer (boşluk kullanılamaz)
assert	else	if	☑ a2	☒ 4ogrenci (sayı ile başlayamaz)
break	except	import	☑ Toplam	☒ *2 (* sembolü kullanılamaz)
class	False	in	☑ Toplam_Brut	☒ öğrenci (Türkçe karakter içeremez)
continue	finally	is	☑ Anahtar_10	☒ class (class ayrılmış kelime olduğu için kullanılamaz)
def	for	lambda		

Şekil 2.7: Python için özel durumlar

Değişkenlere Veri Atama

Python'da değişkenlere değer atamak için = (eşittir) operatörü kullanılır. Bu, değişkenin sol tarafında değişken adını, sağ tarafında ise atanacak değeri belirtir.

Örnekler:

Sayısal Veri Atama (int, float):

```
yas = 25  # tamsayı ataması

sicaklik = 36.6  # ondalıklı sayı ataması
```

Değişkenlere Veri Atama

Python'da değişkenlere değer atamak için = (eşittir) operatörü kullanılır. Bu, değişkenin sol tarafında değişken adını, sağ tarafında ise atanacak değeri belirtir.

Örnekler:

Metinsel Veri Atama (string):

```
isim = "Ali"

sehir = 'Istanbul'
```

Değişkenlere Veri Atama

Python'da değişkenlere değer atamak için = (eşittir) operatörü kullanılır. Bu, değişkenin sol tarafında değişken adını, sağ tarafında ise atanacak değeri belirtir.

Örnekler:

Mantıksal Veri Atama (boolean):

```
aktif = True

mezun = False
```

Değişkenlere Veri Atama

Python'da değişkenlere değer atamak için = (eşittir) operatörü kullanılır. Bu, değişkenin sol tarafında değişken adını, sağ tarafında ise atanacak değeri belirtir.

Örnekler:

Liste Veri Atama (list):

```
sehirler = ["Ankara", "Izmir", "Bursa"]

sayilar = [1, 2, 3, 4, 5]
```

Değişkenlere Veri Atama

Python'da değişkenlere değer atamak için = (eşittir) operatörü kullanılır. Bu, değişkenin sol tarafında değişken adını, sağ tarafında ise atanacak değeri belirtir.

Örnekler:

Sözlük Veri Atama (dict):

```
ogrenci = {"isim": "Ahmet", "yas": 18, "sinif": 12}
```

Değişken ve Sabit Kavramları

Değişken ve sabit, matematik ve programlamada sıkça kullanılan kavramlardır. İkisi arasındaki temel fark, birinin zamanla değişebilmesi, diğerinin ise sabit kalmasıdır.

Değişken:
Tanım: Matematikte ve programlamada bir değişken, belirli bir değeri tutmak üzere tanımlanır, ancak bu değer, zamanla farklı bir değere değişebilir.

Özellikler:
Farklı zamanlarda farklı değerler alabilir.
Örneğin, bir değişken olan x, bir noktada 5, başka bir noktada 10 olabilir.

Değişken ve Sabit Kavramları

Sabit:
Tanım: Sabit ise programın çalışması sırasında ya da matematiksel bir problemde değeri değişmeyen bir semboldür.

Özellikler:Sabit bir değere sahiptir ve bu değer programın akışı boyunca değişmez.

Genellikle programlama dillerinde özel bir anahtar kelime ile tanımlanır ya da isimlendirilirken büyük harfler kullanılır.

```
PI = 3.14159  # PI sabiti değişmez
```

1.4 Bir değişkeni tanımlarken hangi tür veriyi tutacağının belirlenmesi

Değişkenler farklı türlerde verileri (tam sayı, ondalıklı sayı, metin, vb.) tutabilir.

Kullanım amacına göre veri türleri dönüşebilir.

Veri Türü Dönüşümleri

Python'da, bazı durumlarda bir veri türünü başka bir veri türüne dönüştürmek gerekebilir.

Bu işleme veri türü dönüşümü denir.

Python'da veri türü dönüşümleri genellikle iki şekilde yapılır:

1. Otomatik dönüşüm (implicit conversion)
2. Manuel dönüşüm (explicit conversion)

Veri Türü Dönüşümleri

Python'da, bazı durumlarda bir veri türünü başka bir veri türüne dönüştürmek gerekebilir.

Veri türü dönüşümü, programın doğru ve verimli çalışmasını sağlamak için önemlidir, özellikle kullanıcı girdilerinde veya matematiksel işlemler gibi durumlarda dönüşüm kaçınılmazdır.

Python'da veri türü dönüşümleri genellikle iki şekilde yapılır:

1. Otomatik dönüşüm (implicit conversion)
2. Manuel dönüşüm (explicit conversion)

Veri Türü Dönüşümleri

Otomatik Dönüşüm (Implicit Conversion)

Python, belirli durumlarda veri türlerini otomatik olarak dönüştürür. Bu, daha düşük veri türlerini daha yüksek veri türlerine dönüştürerek yapılır. Örneğin, tamsayıları ondalık sayılara otomatik olarak dönüştürebilir. Bu işlem genellikle veri kaybını önlemek için yapılır.

```
sayi1 = 5     # int (tamsayı)
sayi2 = 2.5    # float (ondalıklı sayı)

toplam = sayi1 + sayi2  # Python otomatik
olarak int -> float dönüşümü yapar
print(toplam)          # Çıktı: 7.5
```

Bu örnekte, Python tamsayı olan sayi1'i otomatik olarak ondalıklı sayıya dönüştürür çünkü sayi2 float türündedir ve float ile yapılan işlemler float türünde bir sonuç döner.

Veri Türü Dönüşümleri

Manuel Dönüşüm (Explicit Conversion)

Bazı durumlarda, Python otomatik olarak veri türünü dönüştürmez ve bu dönüşümü manuel olarak yapmanız gerekir. Bu işleme explicit conversion ya da type casting denir. Python'da bazı yerleşik fonksiyonlar kullanılarak veri türü dönüşümü yapılabilir:

int(): Veriyi tamsayıya dönüştürür.
float(): Veriyi ondalıklı sayıya dönüştürür.
str(): Veriyi metin (string) türüne dönüştürür.
bool(): Veriyi mantıksal değere (True/False) dönüştürür.
list(): Veriyi listeye dönüştürür.
tuple(): Veriyi demet (tuple) türüne dönüştürür.

Veri Türü Dönüşümleri

Manuel Dönüşüm Örnekleri:

String'den Sayıya Dönüşüm:

Kullanıcıdan alınan değerler her zaman string (metin) olarak gelir, ancak sayısal işlemler yapabilmek için bu girdileri int veya float türüne dönüştürmek gerekir.

```
yas = input("Yaşınızı giriniz: ")  # Kullanıcıdan alınan veri string türündedir
yas_int = int(yas)  # String'den int'e dönüşüm
print("Girdiğiniz yaşın 5 yıl sonraki hali:", yas_int + 5)
```

Çıktı:
Yaşınızı giriniz: 18
Girdiğiniz yaşın 5 yıl sonraki hali: 23

Veri Türü Dönüşümleri

Manuel Dönüşüm Örnekleri:

Sayısal Değerleri String'e Dönüştürme:

Sayısal verilerle metin birleştirilmek istendiğinde, sayıları metne dönüştürmek gerekir. Python sayısal veri ile string veri türlerini doğrudan birleştiremez.

```
yas = 25
mesaj = "Yaşım " + str(yas)  # int veri str'ye dönüştürülür
print(mesaj)
```

Çıktı:
Yaşım 25

Veri Türü Dönüşümleri

Manuel Dönüşüm Örnekleri:

Ondalıklı Sayıyı Tamsayıya Dönüştürme:

Ondalıklı bir sayıyı tamsayıya dönüştürürken ondalık kısmı kaybedersiniz.

```
ondalik_sayi = 12.56
tamsayi = int(ondalik_sayi)  # float -> int dönüşümü, ondalık kısmı atılır
print(tamsayi)
```

Çıktı:

```
12
```

Veri Türü Dönüşümleri

Manuel Dönüşüm Örnekleri:

Boolean Değer Dönüşümü:

Python'da herhangi bir veri türünü mantıksal (boolean) bir değere dönüştürebilirsiniz. 0, boş string "", None veya boş koleksiyonlar (list, tuple, dict) False olarak değerlendirilir. Diğer tüm değerler True olarak değerlendirilir.

```
sayi = 0
print(bool(sayi))  # Çıktı: False

metin = "Merhaba"
print(bool(metin))  # Çıktı: True
```

Veri Türü Dönüşümleri

Manuel Dönüşüm Örnekleri:

Liste ve Tuple Dönüşümü:

Bir veri yapısını (örneğin listeyi) başka bir veri yapısına (örneğin demet) dönüştürebilirsiniz.

```
liste = [1, 2, 3]
demet = tuple(liste)  # Listeyi tuple'a dönüştürme
print(demet)  # Çıktı: (1, 2, 3)

# Tersi de mümkündür:
yeniden_liste = list(demet)
print(yeniden_liste)  # Çıktı: [1, 2, 3]
```

UYGULAMA 1

- Adınız, soyadınız ve numaranızı değişken(ad,soyad,no) kullanarak ekrana yazdırınız

```
ad= " Selçuk"
soyad= " Bulut"
no= " 1"
print(ad)
print(soyad)
print(tel)
```

UYGULAMA 2

- 1. sınav 100 2. sınav 80 olan notların ortalamasını değişkenler kullanarak hesaplattırınız.

```
sinav1 = 100
sinav2 = 80
ortalama = (sinav1 + sinav2) / 2
print(ortalama)
```

UYGULAMA 3

- Çapı 20 olan dairenin çevresini ve alanını bulan ve ekranda gösteren programı Python ile oluşturalım. *(Daire Ç=2πr A=πr², π=3)*

```
cap = 20
yaricap = cap / 2
pi = 3
cevre = 2 * pi * yaricap
alan = pi * (yaricap ** 2)
print(cevre)
print(alan)
```

UYGULAMA 4

- Maaşı 82324 lira olan birinin %21'lik bir zam ile alacağı yeni maaşı hesaplayan ve ekranda gösteren programı Python ile oluşturalım.

```
maas = 82324
zamOrani = 21
yeni_maas = maas + ((maas * zamOrani) / 100)
print(yeni_maas)
```

OPERATÖRLER

Python'da operatörler, programlarda çeşitli işlemleri gerçekleştirmek için kullanılır.

Operatörler, genellikle iki veya daha fazla değeri alarak işlem yapar ve sonuç üretir.

Operatörler kategorilere ayrılır ve en yaygın olarak kullanılanları **aritmetiksel**, **mantıksal** ve **atama** operatörleridir.

Aritmetiksel operatörler, sayılar üzerinde matematiksel işlemler yapmak için kullanılır.

Operatör	Açıklama	Örnek	Sonuç
+	Toplama	3 + 2	5
-	Çıkarma	5 - 2	3
*	Çarpma	4 * 3	12
/	Bölme	10 / 2	5.0
%	Mod (kalan bulma)	10 % 3	1
**	Üs alma	2 ** 3	8
//	Tamsayı bölmesi (floordiv)	10 // 3	3

+ operatörü

- **Toplama ve birleştirme için kullanılır.**
  print(10+20) print("10"+"20")
- **30** **'1020'**
-
- print(5.5+3.8) print("Bilgisayar"+"Bilimi")
- **9.3** **'Bilgisayar Bilimi'**

- operatörü:

- print(50-30)
- **20**
- print(-7- -9)
- **2**
- print(1.5-0.5)
- **1.0**
- print(4.0-1)
- **3.0**

* operatörü:

- **Çarpma ve stringleri belli sayıda tekrar etmek için kullanılır.**
- print(10*5)
- **50**
- print(-6*-8)
- **48**
- print(1.5*1.5)
- **2.25**

- print("w" * 3)
- www
- print('gol' *5)
- gol gol gol gol gol
- print("uzaksın bana" + " " * 5 + "çook uzak...")
- **uzaksın bana çoook uzak...**

/ operatörü

- **Bölme işlemi gerçekleştirir.**

Sonuç daima float veri türüdür.

- print(21/3)
- **7.0**
- print(int(21.7/3) → **burada veri integer'a dönüştürüldü.**
  7
- print(21/0) →**hata**

// operatörü:

- **Bölme işleminde kalan sayı göz ardı edilir.(Taban Bölme)**
- print(25//6)
- **4**
  print(6//25)
- **0**
  print(4.5//1.2 **)**
- **3.0**
  print(2.1//1**2)**
- **0**

Not:
- eksi ve / bölü işleçlerini karakter dizileri ile birlikte kullanamayız.

Üs Alma

- 5**2
- **25**
- 5**-1
- **0.2**
- 5**0
- **1**

Mod İşlemi

- **Bölme sonucunda kalan sayıyı verir.**
- 25%7
- **4**
- 22%11
- **0**
- 6%25
- **6**
- 0%25
- **0**
  25%0→**hata**

5+4*3/3-9	0.0
15-2**4/2+(-2-2)	3.0
0-9**0	-1
100**0.5/10*2	2.0
10%6-1	3
39//12+12/2	9.0
"10+20+30"	10+20+30
"55"+"55"	5555
"110","110"	110 110
5+5+"5"	hata

5+5+"5"	**hata**
str(12+8)+"0"	**200**
int("56")+int("12")	**68**
int("10"),int("10")	**10 10**
str(1),int("666")	**1 666**
"k"*3+str(5)	**kkk5**
2*"6",66	**66 66**
5+4*10/(4+5-9)	**hata**

Mantıksal operatörler, iki veya daha fazla koşulu karşılaştırmak veya bir koşulun doğruluğunu kontrol etmek için kullanılır.

Genellikle koşul ifadelerinde ve karşılaştırmalarda kullanılır.

Operatör	Açıklama	Örnek	Sonuç
`and`	Tüm koşullar doğruysa True	`True and False`	`False`
`or`	Koşullardan biri doğruysa True	`True or False`	`True`
`not`	Doğruyu yanlışa, yanlışı doğruya çevirir	`not True`	`False`

Mantıksal operatörler Örnek:

```
x = True
y = False

Sonuc = x and y     # False (İkisi de doğru değil)
Sonuc = x or y      # True (Biri doğru)
Sonuc = not x       # False (True'nun tersi)
```

Atama Operatörleri, bir değeri bir değişkene atamak veya mevcut bir değeri güncellemek için kullanılır.

Operatör	Açıklama	Örnek	Sonuç
`=`	Değişkene değer atar	`x = 5`	`x = 5`
`+=`	Değeri toplar ve sonucu atar	`x += 3` (x = x + 3)	`x = 8`
`-=`	Değeri çıkarır ve sonucu atar	`x -= 2` (x = x - 2)	`x = 6`
`*=`	Değeri çarpar ve sonucu atar	`x *= 4` (x = x * 4)	`x = 24`
`/=`	Değeri böler ve sonucu atar	`x /= 2` (x = x / 2)	`x = 12.0`
`%=`	Mod alır ve sonucu atar	`x %= 5` (x = x % 5)	`x = 2.0`
`**=`	Üssünü alır ve sonucu atar	`x **= 3` (x = x ** 3)	`x = 8`
`//=`	Tamsayı bölümünü alır ve sonucu atar	`x //= 3` (x = x // 3)	`x = 2`

Karşılaştırma Operatörleri, iki değeri karşılaştırmak ve bu karşılaştırmanın sonucunda True veya False döndürmek için kullanılır. Bu operatörler genellikle koşul ifadelerinde ve döngülerde kullanılır.

Operatör	Açıklama	Örnek	Sonuç
==	Eşit mi?	5 == 5	True
!=	Eşit değil mi?	5 != 3	True
>	Büyük mü?	5 > 3	True
<	Küçük mü?	3 < 5	True
>=	Büyük veya eşit mi?	5 >= 5	True
<=	Küçük veya eşit mi?	3 <= 5	True

Örneklerle Karşılaştırma Operatörleri:

```
x = 10
y = 10
print(x == y)  # True (x, y'ye eşit)
```

```
x = 10
y = 5
print(x != y)  # True (x, y'ye eşit değil)
```

```
x = 10
y = 5
print(x > y)  # True (x, y'den büyük)
```

```
x = 3
y = 7
print(x < y)  # True (x, y'den küçük)
```

Örneklerle Karşılaştırma Operatörleri:

```
x = 10
y = 10
print(x >= y)  # True (x, y'ye eşit veya büyük)
```

```
x = 7
y = 7
print(x <= y)  # True (x, y'ye eşit veya küçük)
```

Karşılaştırma Operatörleri Kullanım Örneği:

```
x = 15
y = 20

if x < y:
   print("x, y'den küçüktür")  # Bu blok çalışır
else:
   print("x, y'den büyük veya eşittir")
```

Çıktı:
x, y'den küçüktür

Karşılaştırma Operatörlerinin Zincirlenmesi:

Python'da birden fazla karşılaştırma operatörü aynı ifadede zincirlenebilir:

```
x = 10
print(5 < x < 15)  # True (x hem 5'ten büyük hem de 15'ten küçük)
```

İşlem Önceliği:

İşlem önceliği, operatörlerin hangi sırayla değerlendirileceğini belirler. Örneğin, çarpma ve bölme işlemleri toplama ve çıkarma işlemlerinden önce yapılır.

Python'da İşlem Önceliği Sırası:

1. Parantezler ()
2. Üs alma **
3. İşaret (artı/eksi)
4. Çarpma, bölme, mod, tamsayı bölmesi *, /, %, //
5. Toplama ve çıkarma +, -
6. Karşılaştırma >, <, >=, <=
7. Mantıksal NOT not
8. Mantıksal AND and
9. Mantıksal OR or
10. Atama =, +=, -=, vb.

Örneklerle İşlem Önceliği:

Parantezlerin Kullanımı

Parantezler işlem önceliğini değiştirmek için kullanılır. İçerideki işlemler önce yapılır.

```
sonuc = 5 + 3 * 2
print(sonuc)  # Çıktı: 11 (Çarpma önce yapılır: 3 * 2 = 6, sonra 5 + 6 = 11)

sonuc = (5 + 3) * 2
print(sonuc)  # Çıktı: 16 (Parantez içindeki işlem önce yapılır: 5 + 3 = 8, sonra 8 * 2 = 16)
```

Örneklerle İşlem Önceliği:

Üs Alma (**) Önceliği
Üs alma işlemi çarpma ve bölmeden daha önce yapılır.

```
sonuc = 2 + 3 ** 2
print(sonuc)  # Çıktı: 11 (3 ** 2 = 9, sonra 2 + 9 = 11)

sonuc = (2 + 3) ** 2
print(sonuc)  # Çıktı: 25 (Parantez içindeki işlem önce yapılır: 2 + 3 = 5, sonra 5 ** 2 = 25)
```

Örneklerle İşlem Önceliği:

Çarpma ve Bölme Önceliği
Çarpma, bölme, mod ve tamsayı bölmesi, toplama ve çıkarmadan önce yapılır.

```
sonuc = 10 - 4 / 2 + 3 * 2
print(sonuc)  # Çıktı: 13 (4 / 2 = 2, 3 * 2 = 6, sonra 10 - 2 + 6 = 13)

sonuc = (10 - 4) / (2 + 3) * 2
print(sonuc)  # Çıktı: 2.4 (Parantez içindeki işlemler önce yapılır: 10 - 4 = 6, 2 + 3 = 5, sonra 6 / 5 * 2 = 2.4)
```

Örneklerle İşlem Önceliği:

Mantıksal Operatörlerde Öncelik
Mantıksal NOT (not), AND (and) ve OR (or) operatörlerinin farklı öncelikleri vardır. not en yüksek, ardından and, en sonunda or gelir.

```
sonuc = True or False and False
print(sonuc)  # Çıktı: True (AND işlemi önce yapılır: False and False = False, sonra True or False = True)

sonuc = (True or False) and False
print(sonuc)  # Çıktı: False (Parantez içindeki işlem önce yapılır: True or False = True, sonra True and False = False)
```

Örneklerle İşlem Önceliği:

Üs Alma ile İşaretlerin Kullanımı
Üs alma işlemi sağdan sola bağlanır, işaret operatörleriyle birleştiğinde farklı sonuçlar verebilir.

```
sonuc = -3 ** 2
print(sonuc)  # Çıktı: -9 (Üs alma işlemi önce yapılır: 3 ** 2 = 9, sonra -9)

sonuc = (-3) ** 2
print(sonuc)  # Çıktı: 9 (Parantez içindeki işlem önce yapılır: -3 ** 2 = 9)
```

Örneklerle İşlem Önceliği:

İşlem Önceliği ile Karşılaştırma Operatörleri
Karşılaştırma operatörleri, aritmetik operatörlerden sonra gelir.

```
sonuc = 3 + 5 > 7
print(sonuc)  # Çıktı: True (Önce 3 + 5 = 8 yapılır, sonra 8 > 7 kontrol edilir: True)

sonuc = 3 + (5 > 7)
print(sonuc)  # Çıktı: 3 (5 > 7 = False (0), sonra 3 + 0 = 3)
```

Merhaba Python!

```
isim = " Python"

print("Merhaba", isim, end="!\n")
```

Python'da kullanıcıdan herhangi bir veri alıp, yazdığımız programları tek taraflı olmaktan kurtarmak için

Bu programı çalıştırdığımızda söyle bir çıktı alacağımızı biliyorsunuz:

input()

adlı bir fonksiyondan faydalanıyoruz.

input()

input() daha önce ögrendigimiz **type(), len() ve print()** gibi bir **fonksiyondur.**

input() Kullanıcıdan bilgi almak için kullanılır.

input() Alınan bilgilerin türü karakter yani stringdir.

Ör: Kullanıcıya ismini sorup "Merhaba İsim" yazdıran kodlar.

```
input("İsminiz nedir? ")

isim=input("İsminiz nedir? ")
print("Merhaba", isim)
```

İsminiz nedir?

Merhaba Selçuk

Kullanıcın girdiği iki sayıyı toplayan kodlar:

```
sayi1 = int(input("ilk sayıyı giriniz: "))

sayi2 = int(input("ikinci sayıyı giriniz: "))

toplam = sayi1 + sayi2

print(sayi1, "+", sayi2, "=", toplam)
```

Girilen yaşa yorum yazan kodlar:

```
yas =input("Yaşınızı giriniz : ")

print("Demek", yas, "yaşındasın.")

print("Daha çok gençsin")
```

Saniyeyi dakikaya çeviren kodlar:

```
saniye=int(input("Saniye sayısını giriniz:"))

dakika=saniye/60

print(saniye,"saniye",dakika,"dakika eder")
```

```
print("merhaba.")
input()
print("ismini söyler misin...?")
a=input()
print(a,"yaşın kaç...?")
input()
print(a,"okulun...?")
input()
print(a,"memlektin...?")
input( )
print( a," kaçını sınıfsın ")
input( )
```

Kullanıcının girdiği iki notun ortalamasını bulan programın python kodlarını yazınız

Input Fonksiyonu Örnekleri

- Kullanıcıya adını, doğalgaz ilk ve son endeksleri sorunuz. Son endeksten ilk endeksi çıkararak kaç metreküp harcandığını bulunuz. Harcanan miktarı 4.08 ile çarparak tüketim bedelini hesaplayınız. Şu şekilde çıktı veriniz:

Sayın ....., bu ay ..... metreküp doğalgaz harcadınız.
Ödemeniz gereken tutar ..... TL'dir.