

2024-2025 EĐİTİM VE ÖĐRETİM YILI
PROGRAMLAMA DİLLERİ MODÜLÜ (PYTHON)

1. ÜNİTE

PYTHON PROGRAMLAMAYA GİRİŐ

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.

```
file = None
fingerprints = set()
logdupes = True
debug = debug
logger = logging.getLogger(__name__)
path:
self.file = open(os.path.join(settings.get('job_dir', 'logs'), 'fingerprints.log'), 'a')
self.file.seek(0)
self.fingerprints.update(request.fingerprints)

def __init__(cls, settings):
    debug = settings.getbool('debug', False)
    return cls(job_dir(settings), debug)

def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + os.linesep)

def request_fingerprint(self, request):
    return request_fingerprint(request)
```

```
file = None
fingerprints = set()
logdupes = True
debug = debug
logger = logging.getLogger(__name__)
path:
self.file = open(os.path.join(
self.file.seek(0)
self.fingerprints.update(re

method
m_settings(cls, settings):
bug = settings.getbool("supers
return cls(job_dir(settings), de

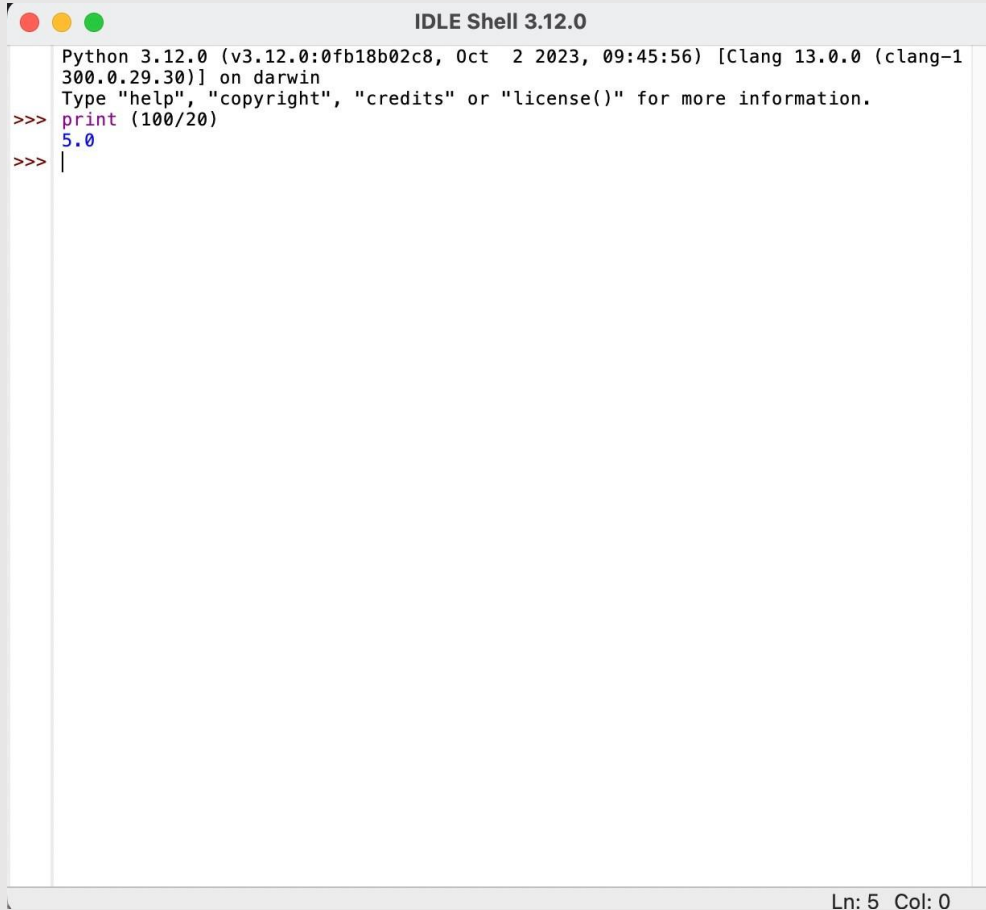
request_seen(self, request):
p = self.request_fingerprint(re
if fp in self.fingerprints:
return True
self.fingerprints.add(fp)
if self.file:
self.file.write(fp + os.lin

request_fingerprint(self, requ
return request_fingerprint(re
```

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 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print (100/20)
5.0
>>> |
```

Ln: 5 Col: 0

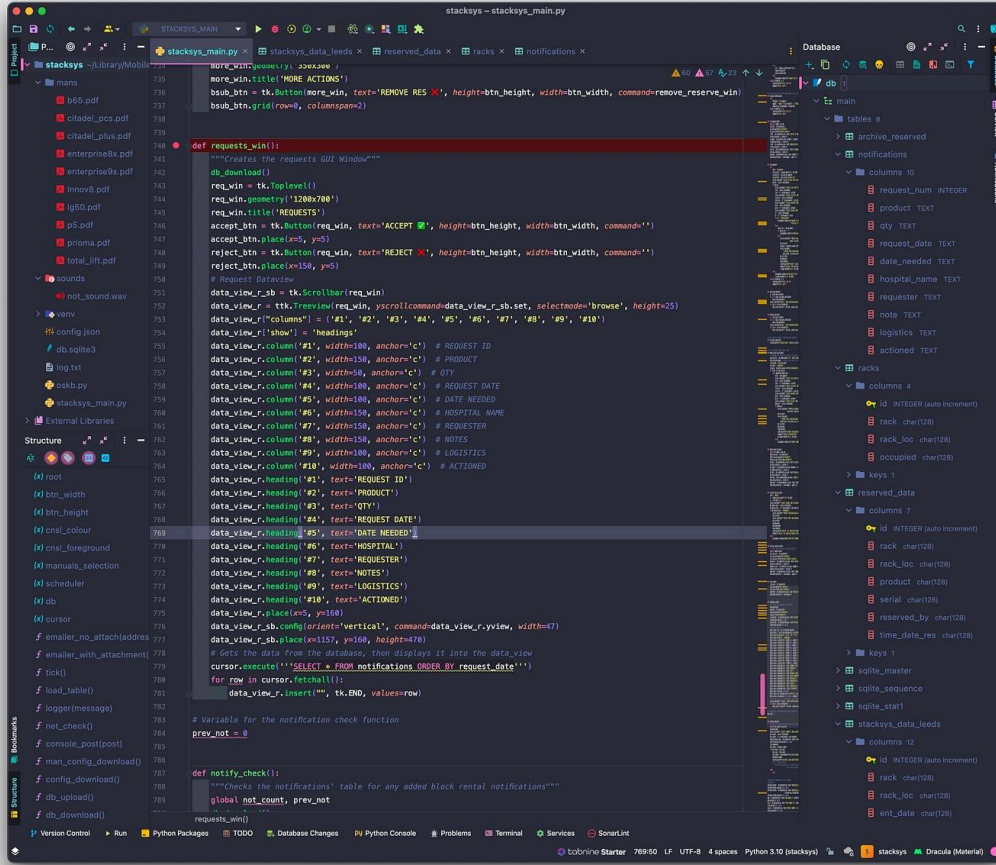
Python IDLE

- 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.



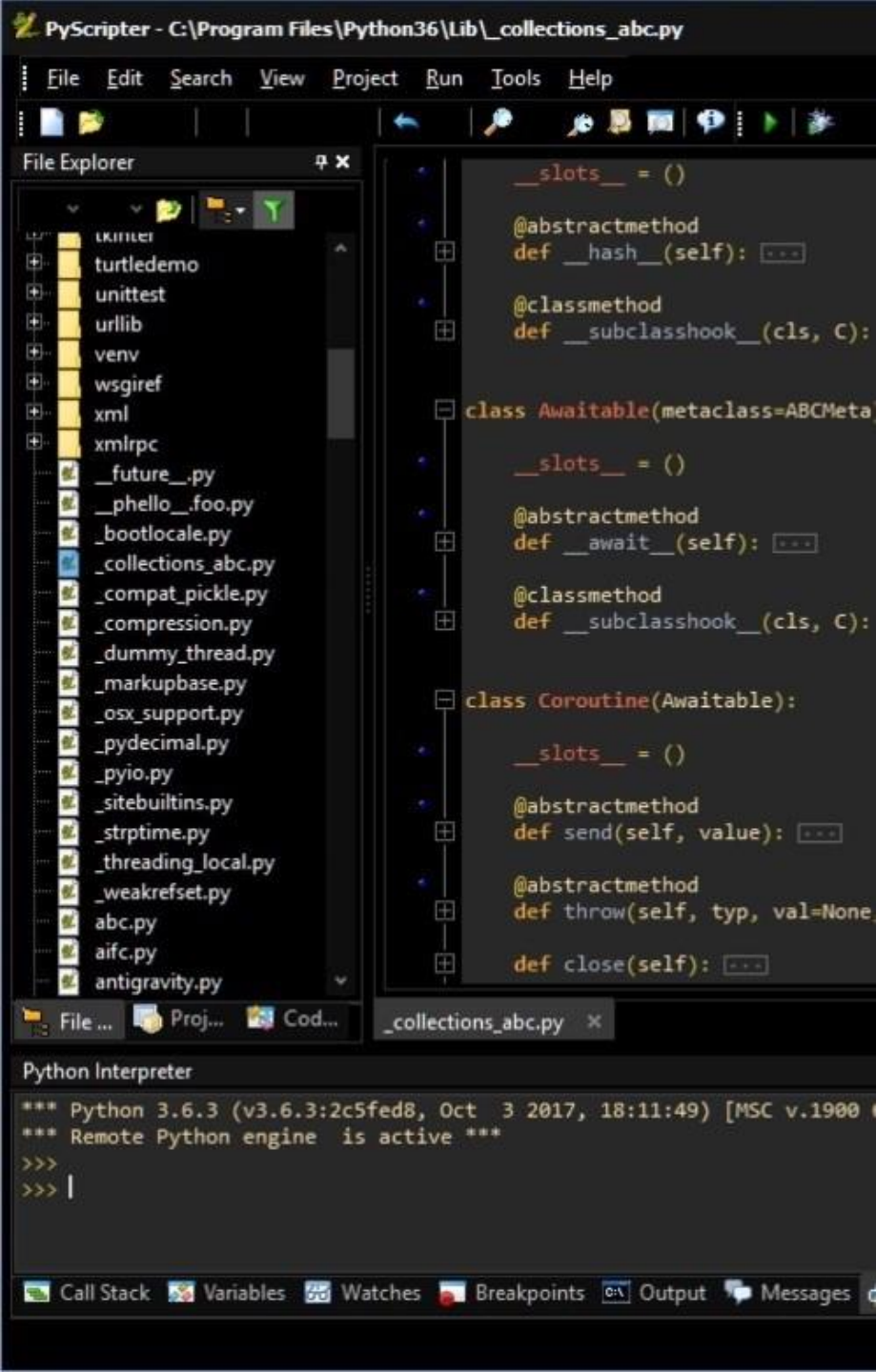
PyCharm

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

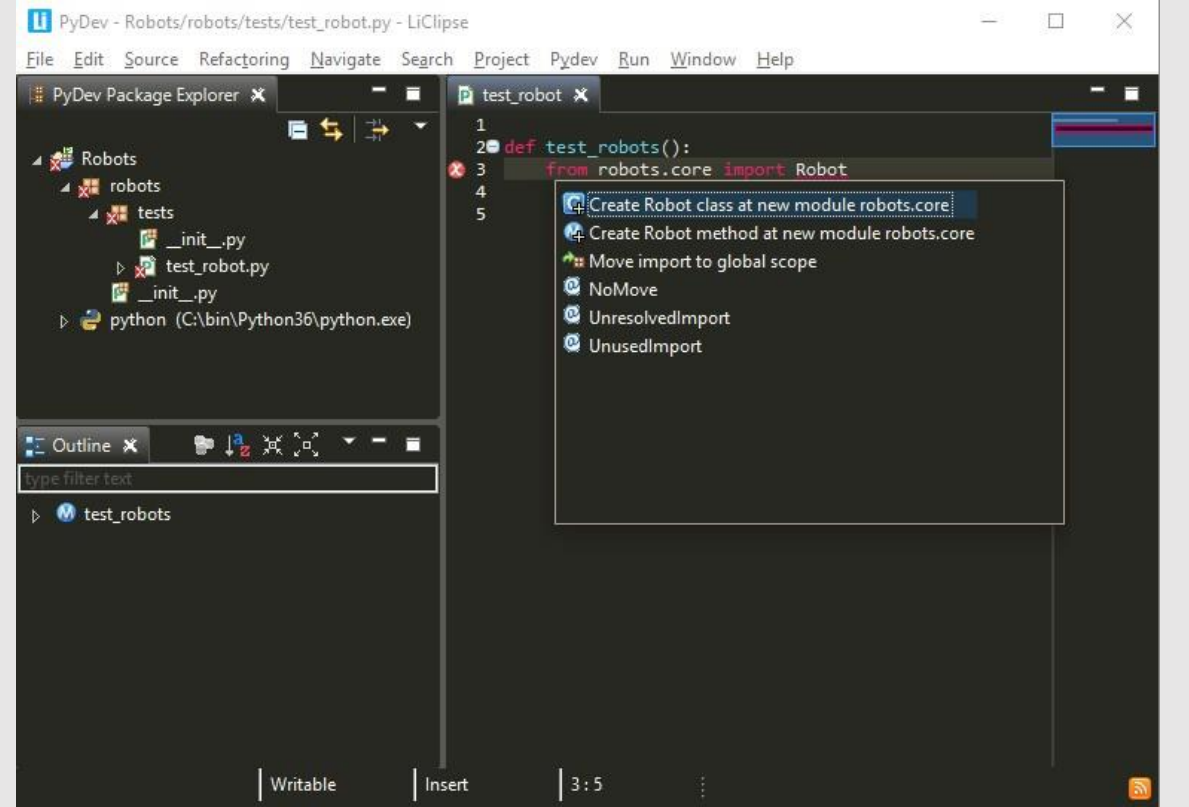
PyDev (Eclipse ile birlikte)

Avantajlar:

- Eclipse, yazılım geliřtiricilerin diđer programlama dillerinde yazılmıř kodları geliřtirmelerine ve test etmelerine olanak sađlayan eklentileriyle bilinen ücretsiz, Java tabanlı bir geliřtirme platformudur.
- Eclipse ile entegre, çoklu dil desteđi (Java, C++, vb.)
- Geliřmiř hata ayıklama ve otomatik tamamlama özellikleri
- Ücretsiz

Dezavantajlar:

- Eclipse'in karmařık yapısı nedeniyle bařlangıç için öğrenmesi zor olabilir
- Ađır ve yavař olabilir



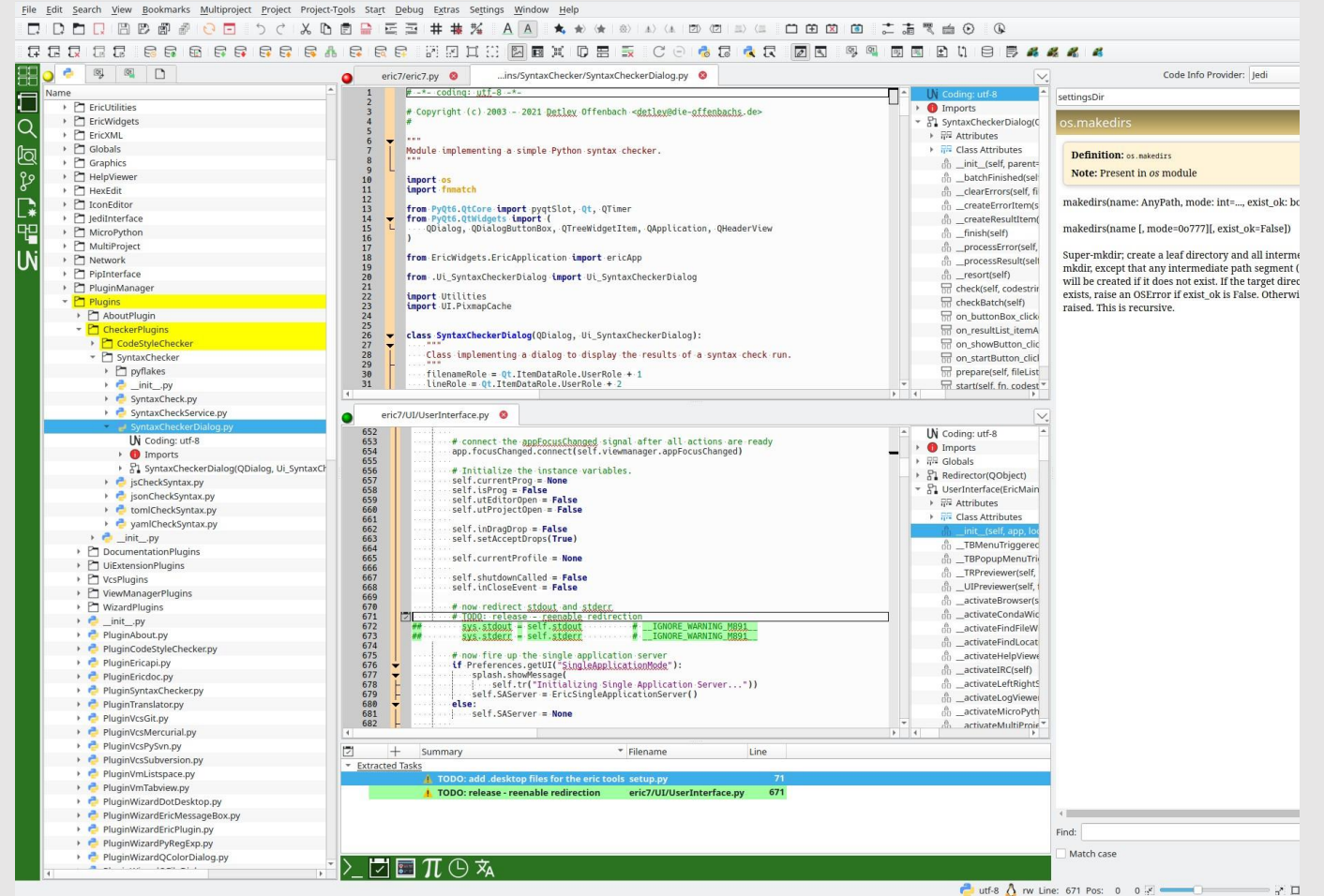
Eric Python

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



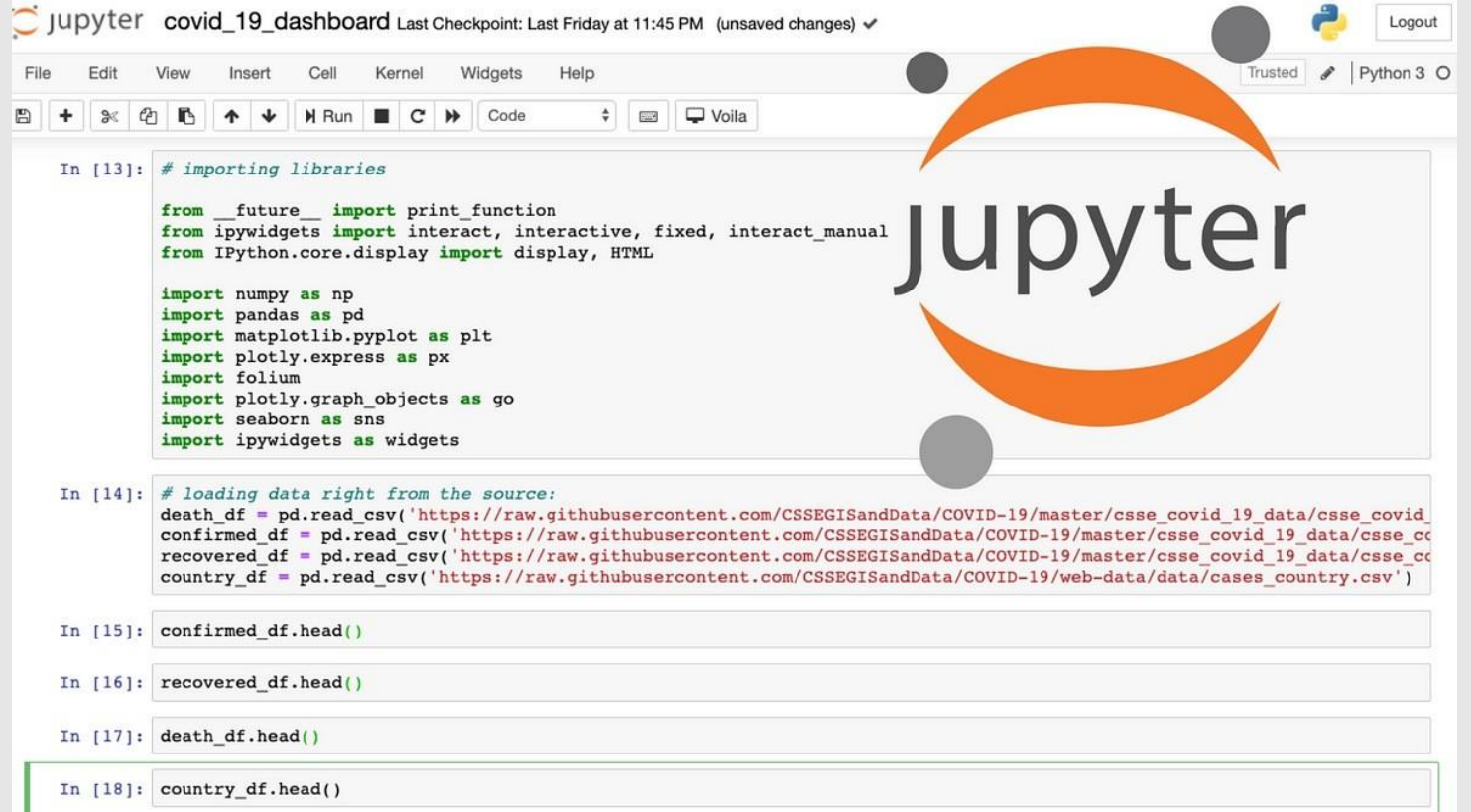
Jupyter Notebook

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



The screenshot shows a Jupyter Notebook interface with the title 'covid_19_dashboard'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and other actions. The main area contains a code cell with the following code:

```
In [13]: # importing libraries

from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
from IPython.core.display import display, HTML

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import folium
import plotly.graph_objects as go
import seaborn as sns
import ipywidgets as widgets
```

Below the code cell, there are four more code cells, each followed by a blank output area:

```
In [14]: # loading data right from the source:
death_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_data/confirmed_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_cc
recovered_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_cc
country_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/web-data/data/cases_country.csv')
```

```
In [15]: confirmed_df.head()
```

```
In [16]: recovered_df.head()
```

```
In [17]: death_df.head()
```

```
In [18]: country_df.head()
```

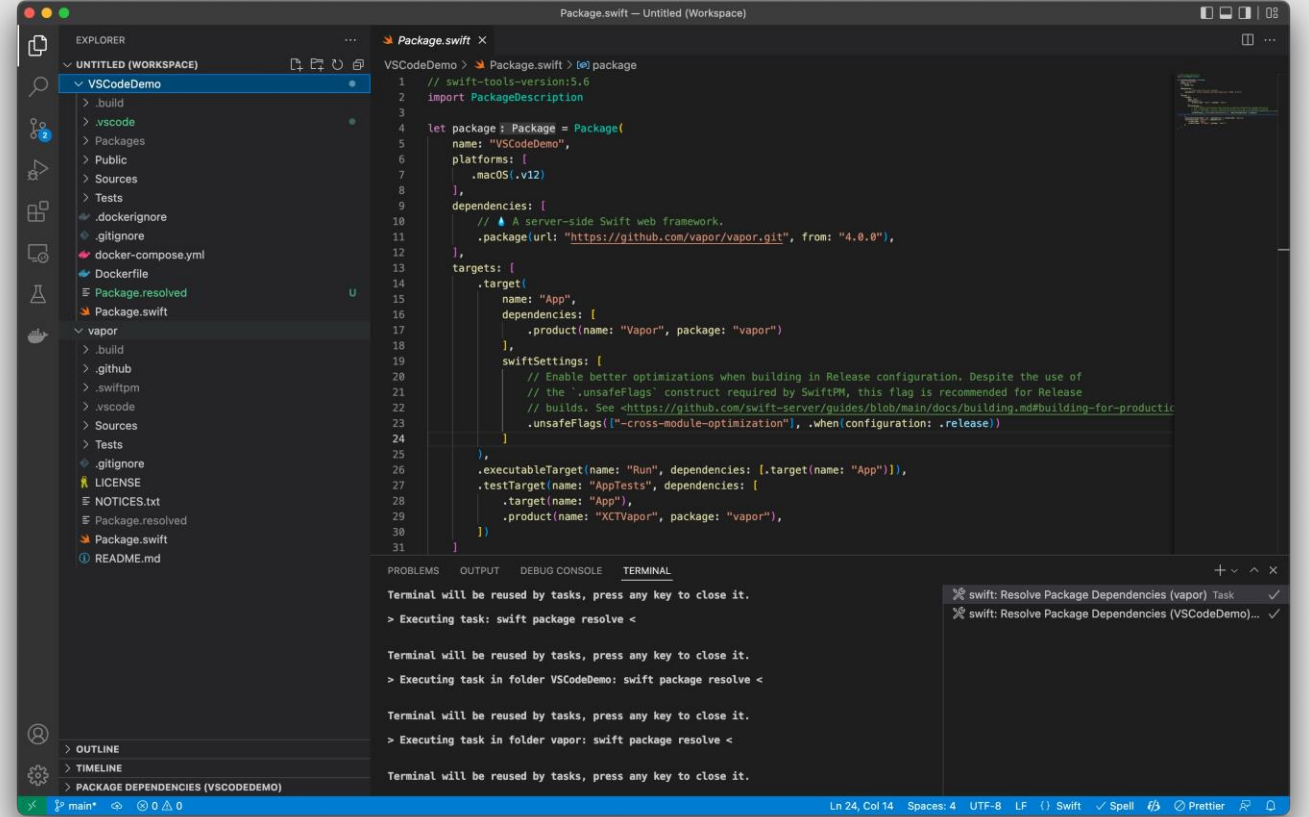
Visual Studio Code (VS Code)

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



The screenshot shows the Visual Studio Code editor interface. The Explorer view on the left shows a project structure with folders like .build, .vscode, Packages, Public, Sources, Tests, .dockerignore, .gitignore, docker-compose.yml, Dockerfile, Package.resolved, Package.swift, and vapor. The Package.swift file is open in the editor, showing Swift code for a package named 'VSCoDemo'. The code includes dependencies on 'vapor' and defines targets for 'App' and 'Run'. The Terminal view at the bottom shows the output of the 'swift package resolve' command, indicating that the terminal will be reused by tasks.

```
1 // swift-tools-version:5.6
2 import PackageDescription
3
4 let package : Package = Package(
5     name: "VSCoDemo",
6     platforms: [
7         .macOS(.v12)
8     ],
9     dependencies: [
10        // A server-side Swift web framework.
11        .package(url: "https://github.com/vapor/vapor.git", from: "4.0.0"),
12    ],
13    targets: [
14        .target(
15            name: "App",
16            dependencies: [
17                .product(name: "Vapor", package: "vapor")
18            ],
19            swiftSettings: [
20                // Enable better optimizations when building in Release configuration. Despite the use of
21                // the ".unsafeFlags" construct required by SwiftPM, this flag is recommended for Release
22                // builds. See <https://github.com/swift-server/guides/blob/main/docs/building.md#building-for-productic
23                .unsafeFlags(["-cross-module-optimization"], .when(configuration: .release))
24            ]
25        ),
26        .executableTarget(name: "Run", dependencies: [.target(name: "App")]),
27        .testTarget(name: "AppTests", dependencies: [
28            .target(name: "App"),
29            .product(name: "XCTVapor", package: "vapor"),
30        ])
31    ]
32 )
```

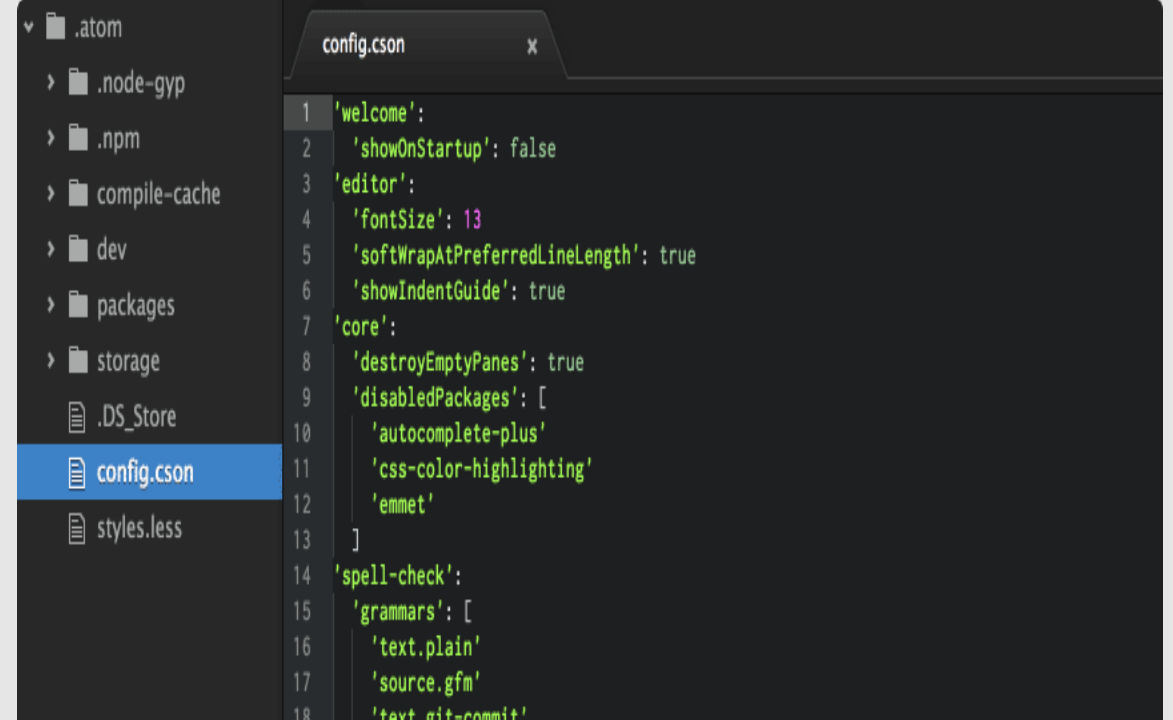
Atom

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



```
1 'welcome':
2 'showOnStartup': false
3 'editor':
4 'fontSize': 13
5 'softWrapAtPreferredLineLength': true
6 'showIndentGuide': true
7 'core':
8 'destroyEmptyPanels': true
9 'disabledPackages': [
10 'autocomplete-plus'
11 'css-color-highlighting'
12 'emmet'
13 ]
14 'spell-check':
15 'grammars': [
16 'text.plain'
17 'source.gfm'
18 'text.git-commit'
```



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:

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.



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:

- Ücretli:** Sublime Text ücretsiz deneme sürümü sunsa da, tam sürümünü kullanmak için bir **lisans** satın almak gerekiyor.
- IDE Özellikleri Zayıf:** Sublime Text, tam anlamıyla bir **IDE** değildir. Derleme, hata ayıklama ve test gibi gelişmiş özellikler için eklentilere bağımlıdır.
- 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.

```
1 """
2 Plot a terrain model and a polar plot side by side.
3 """
4
5 # Third party imports
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import matplotlib.cm
9 import matplotlib.colors
10 import mpl_toolkits.mplot3d # pylint: disable=unused-import
11
12 plt.style.use("dark_background")
13
14 def generate_polar_plot():
15     """Generate an example polar slice plot."""
16     # Compute pi slices
17     n_slices = 20
18     theta = np.linspace(0, 2 * np.pi, n_slices, endpoint=False)
19     radii = 10 * np.random.rand(n_slices)
20     width = np.pi / 4 * np.random.rand(n_slices)
21     fig, ax = plt.subplots(figsize=(8, 3))
22     fig.patch.set_facecolor('#959595')
23     ax = plt.subplot(1, 2, projection='polar')
24     ax.set_facecolor('#959595')
25     bars = ax.bar(theta, radii, width=width, bottom=0)
26
27 # Use custom colors and opacity
28 for radius, plot_bar in zip(radii, bars):
29     plot_bar.set_facecolor(plt.cm.viridis(radius / 10.))
30     plot_bar.set_alpha(0.5)
31
32 def generate_dem_plot():
33     """Generate a 3D representation of a terrain DEM."""
34     dem_path = 'jacksboro_fault_dem.zip'
35     with np.load(dem_path) as dem:
36         z_data = dem['elevation']
37         n_rows, n_cols = z_data.shape
38         x_data = np.linspace(dem['xmin'], dem['xmax'], n_cols)
39         y_data = np.linspace(dem['ymin'], dem['ymax'], n_rows)
40         x_data, y_data = np.meshgrid(x_data, y_data)
41
42 region = np.s_[5:50, 5:50]
43 x_region, y_region, z_region = (
44     x_data[region], y_data[region], z_data[region])
45
46 axes = plt.subplot(1, 2, 1, projection='3d')
47 axes.set_facecolor('#959595')
48 plt.locator_params(axis='y', nbins=6)
49 plt.locator_params(axis='x', nbins=6)
50
51 light_source = matplotlib.colors.LightSource(270, 45)
52 # To use a custom lighting mode, override the built-in shading and pass
53 # in the rgb colors of the shaded surface calculated from "shade".
54 rgb_map = light_source.shade(z_data, cmap=matplotlib.cm.gist_earth,
55                             rgba_params=matplotlib.rcParams['cmap_d'])
```

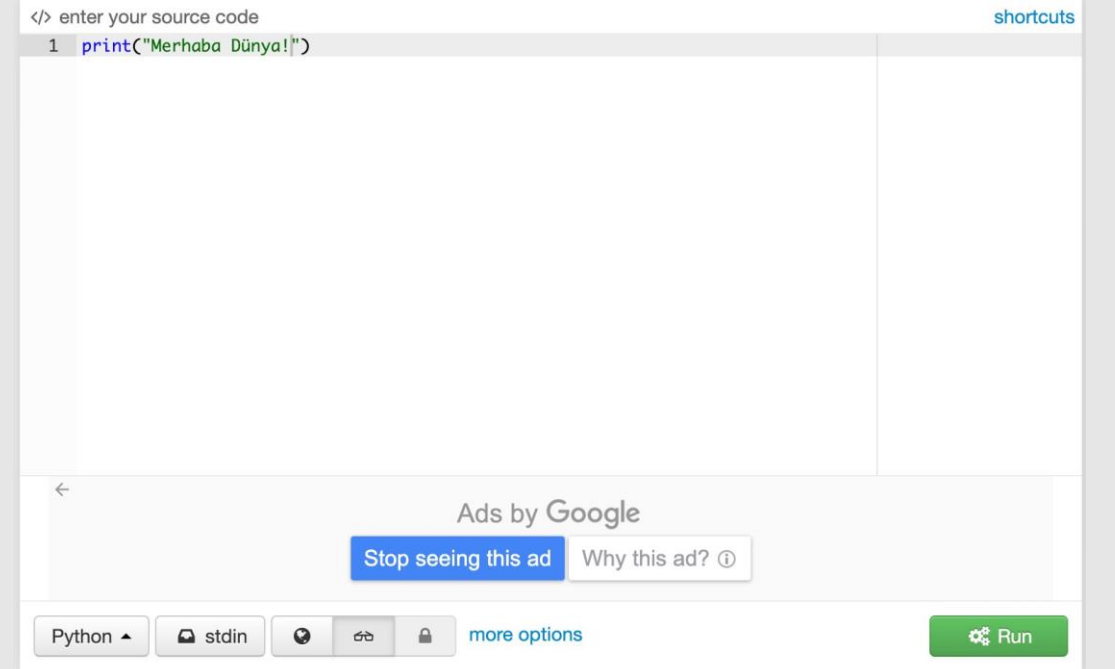
```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Mon May 4 17:07:07 2020
5
6 @author: juanis
7 """
8
9 # pylint: disable=invalid-name
10
11 # %% Imports
12 import numpy as np
13 import matplotlib.pyplot as plt
14 import pandas as pd
15 import time
16 from scipy.spatial import KDTree, Voronoi, voronoi_plot_2d
17 from shapely.geometry import Point, Polygon
18 plt.style.use("dark_background")
19
20 file_paths = ["/Users/juanitagomez/Documents/SpyderDocs/airports_CO.dat",
21              "/Users/juanitagomez/Documents/SpyderDocs/borders_CO.dat"]
22
23 airports_CO = file_paths[0]
24 borders_CO = file_paths[1]
25
26 # %% Read files
27 airports_CO = pd.read_csv(airports_CO, sep="|", names=["coord-y", "coord-x",
28              "Altitud", "City", "Department", "Airport"])
29 borders_CO = pd.read_csv(borders_CO, sep="|", names=["coord-y", "coord-x"])
30
31 class FlightOperations:
32     def __init__(self, airports, borders):
33         self.airports = airports
34         self.borders = borders
35         self.points = self.airports[['coord-x', 'coord-y']].to_numpy()
36         self.hull = ConvexHull(self.points)
37         self.vor = Voronoi(self.points)
38
39     def sleep_wrapper(self):
40         time.sleep(0.003)
41
42     def plotAirports(self):
43         """ Plot map with airports """
44         voronoi_plot_2d(self.vor)
45         plt.plot(self.borders['coord-x'], self.borders['coord-y'])
46         plt.show()
47
48     def findNearestPointID(self, point):
49         """ Find nearest airport given a point in any location using KDTree """
50         points = self.airports[['coord-x', 'coord-y']].to_numpy()
```

Python IDE türleri

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

Örnek online IDE'ler

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



```
</> enter your source code shortcuts
1 print("Merhaba Dünya!")
```

← Ads by Google
Stop seeing this ad Why this ad? ⓘ

Python ▾ stdin [] [] more options Run

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

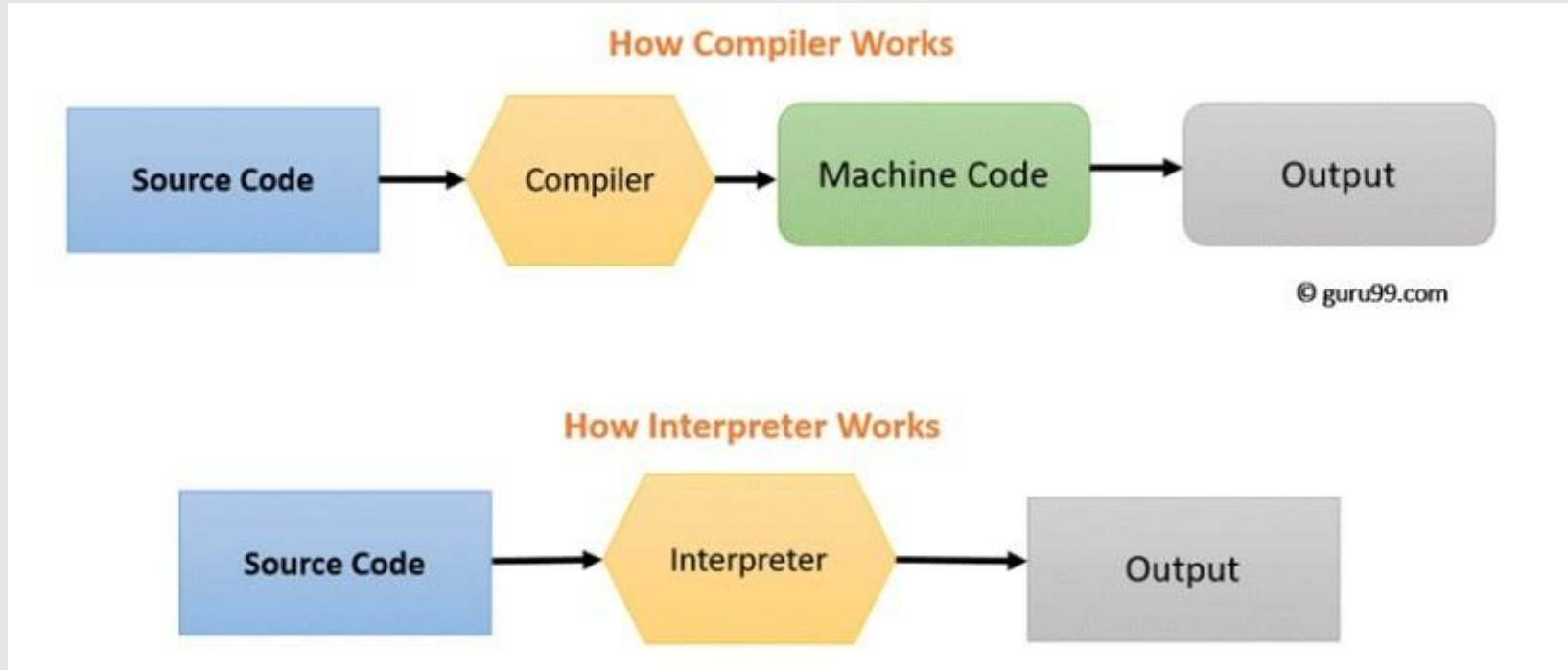
- Python
- JavaScript
- Ruby
- PHP



Derlenen ve Yorumlanan Diller Arasındaki Farklar:

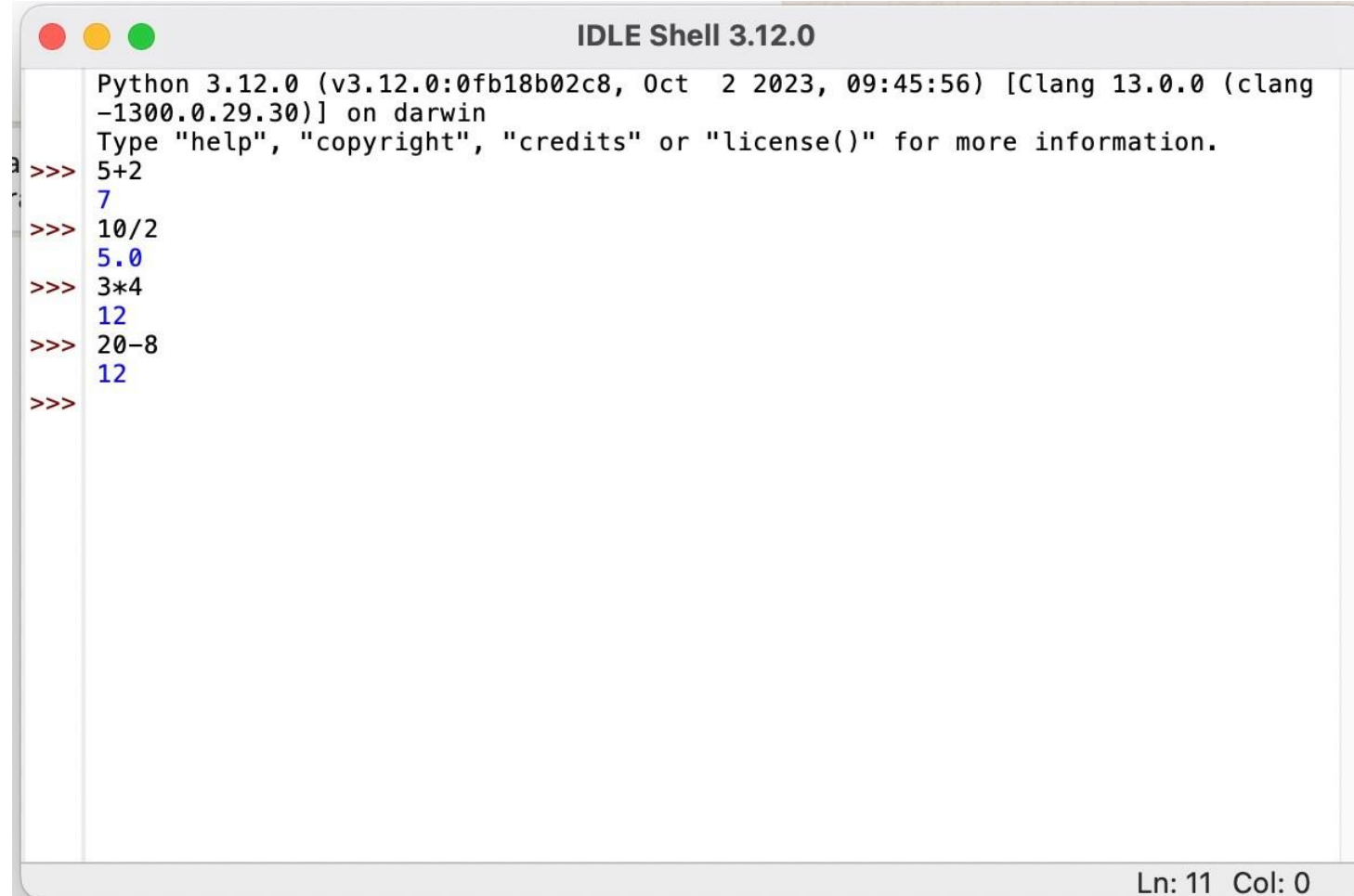
- 1. Çalıştırma Şekli:** Derlenen dillerde kod önce makine diline çevrilir ve bu şekilde çalıştırılır. Yorumlanan dillerde ise kod doğrudan satır satır çalıştırılır.
- 2. Hız:** Derlenen diller genellikle daha hızlı çalışır çünkü doğrudan makine dilinde çalıştırılırlar. Yorumlanan diller ise her satırı 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.



```
Python 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> 5+2
7
>>> 10/2
5.0
>>> 3*4
12
>>> 20-8
12
>>>
```

Ln: 11 Col: 0

Yorum Satırları

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

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

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.

#

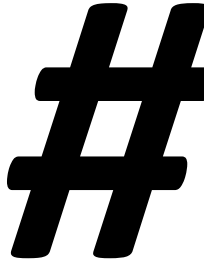
ÖRNEK>

```
*IDLE Shell 3.12.0*
Python 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
... # Bu bir tek satır yorumdur
... print("Merhaba Dünya") # Bu da satır sonu yorumudur
... |
Ln: 3 Col: 0
```

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

```
*IDLE Shell 3.12.0*
Python 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> # Bu bir
... # çok satırlı
... # yorumdur
... print("Merhaba Dünya")
Ln: 6 Col: 22
```

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

```
*IDLE Shell 3.12.0*
Python 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> """
... Bu bir
... çok satırlı
... docstring yorumudur
... """
... print("Merhaba dünya")
Ln: 8 Col: 5
```

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.

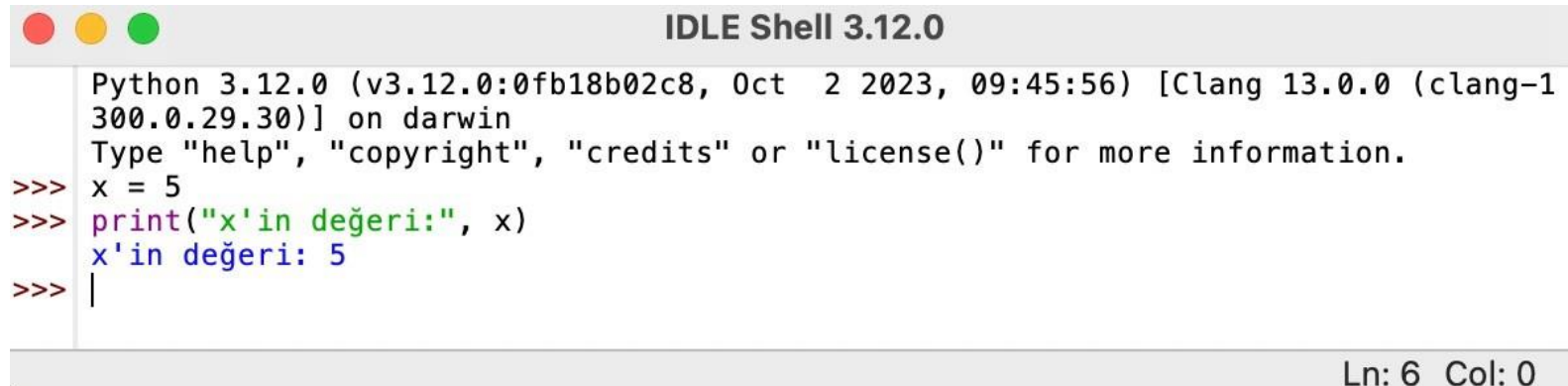
print() fonksiyonu

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:



```
Python 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> x = 5
>>> print("x'in değeri:", x)
x'in değeri: 5
>>> |
```

Ln: 6 Col: 0

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.

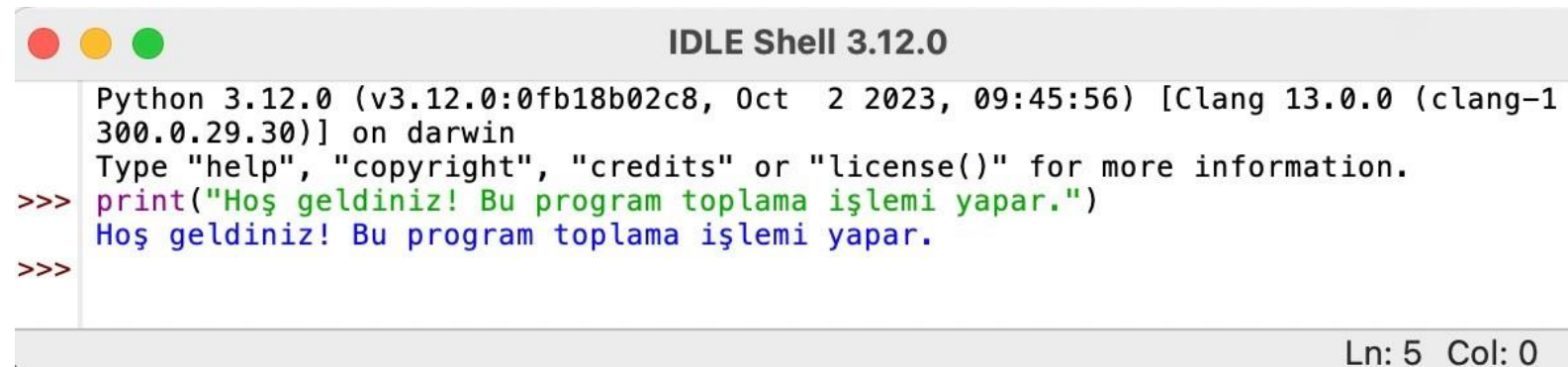
print() fonksiyonu

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:



```
Python 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hoş geldiniz! Bu program toplama işlemi yapar.")
Hoş geldiniz! Bu program toplama işlemi yapar.
>>>
```

Ln: 5 Col: 0

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.

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.

```
*untitled*  
def kare(x):  
    return x * x  
  
print(kare(4)) # Doğru sonucu yazdırır: 16  
Ln: 5 Col: 0
```

input() fonksiyonu

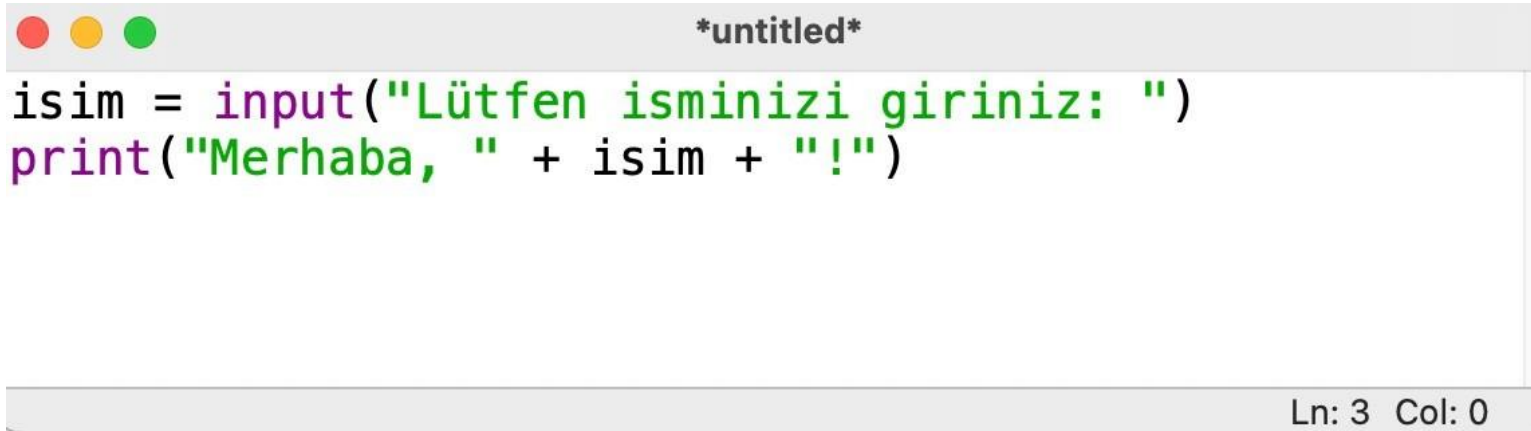
Python'da, kullanıcıdan veri almak için **input()** fonksiyonu kullanılır.

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

input() fonksiyonu

Basit Bir Kullanıcı Girişi

Kullanıcıdan isim almak ve ekrana yazdırmak:



```
isim = input("Lütfen isminizi giriniz: ")
print("Merhaba, " + isim + "!")
```

Ln: 3 Col: 0

Çıktı:

Lütfen isminizi giriniz: Onur
Merhaba, Onur!

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("10 yıl sonra" , yas+10 )
```

Çıktı:

Kaç yaşındasınız? 15

10 yıl sonra 25 yaşında olacaksınız.

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(ad, yas, "yaşındasınız.")
```

Çıktı:

Adınızı giriniz: Duru

Yaşınızı giriniz: 17

Duru 17 yaşındasınız.

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

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!

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}`

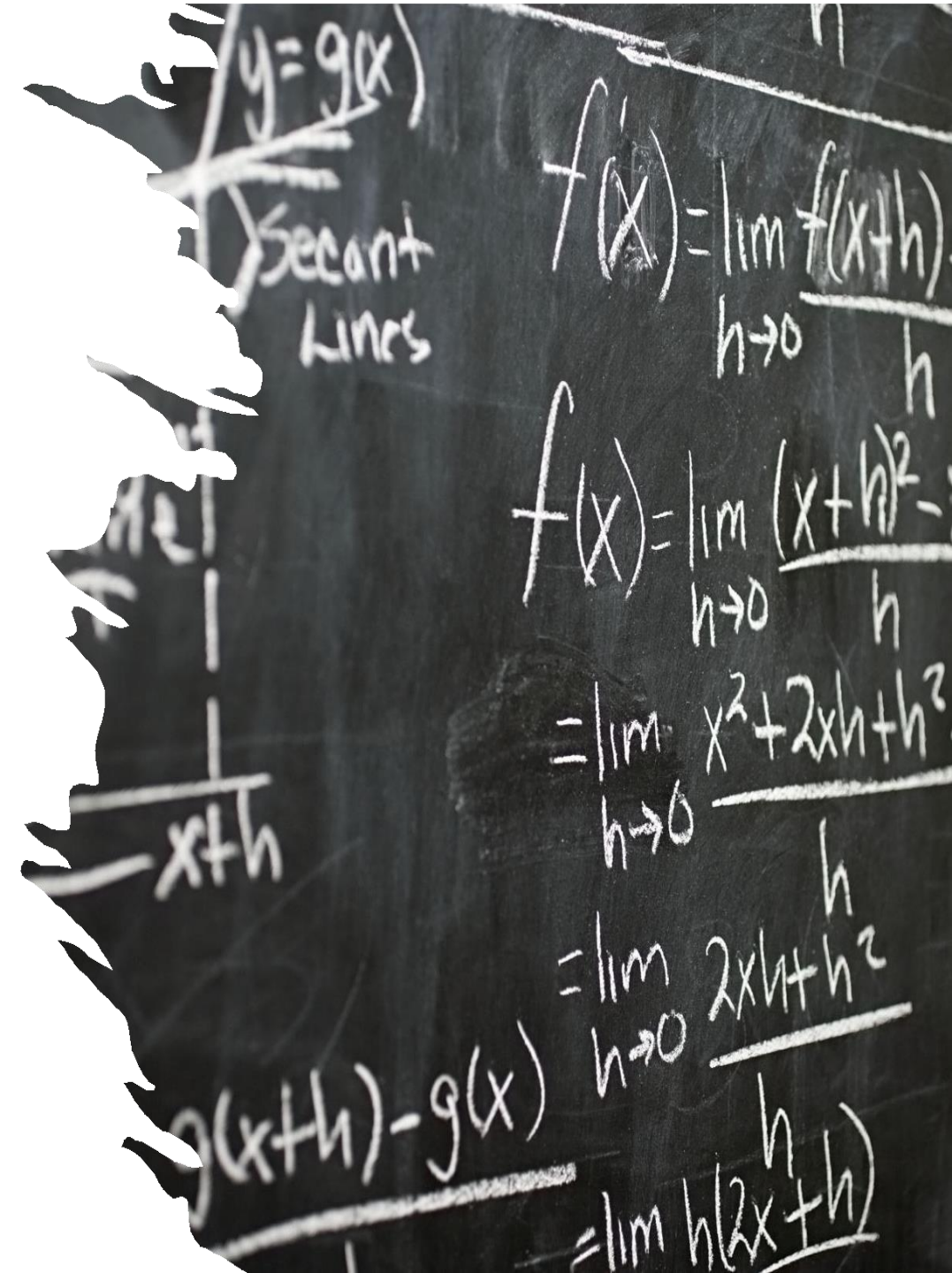
Değişken Tanımlama Kuralları:

İsmlendirme:

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

Doğru: isim, _degisken, yas_23

Yanlış: 23yas, -ad



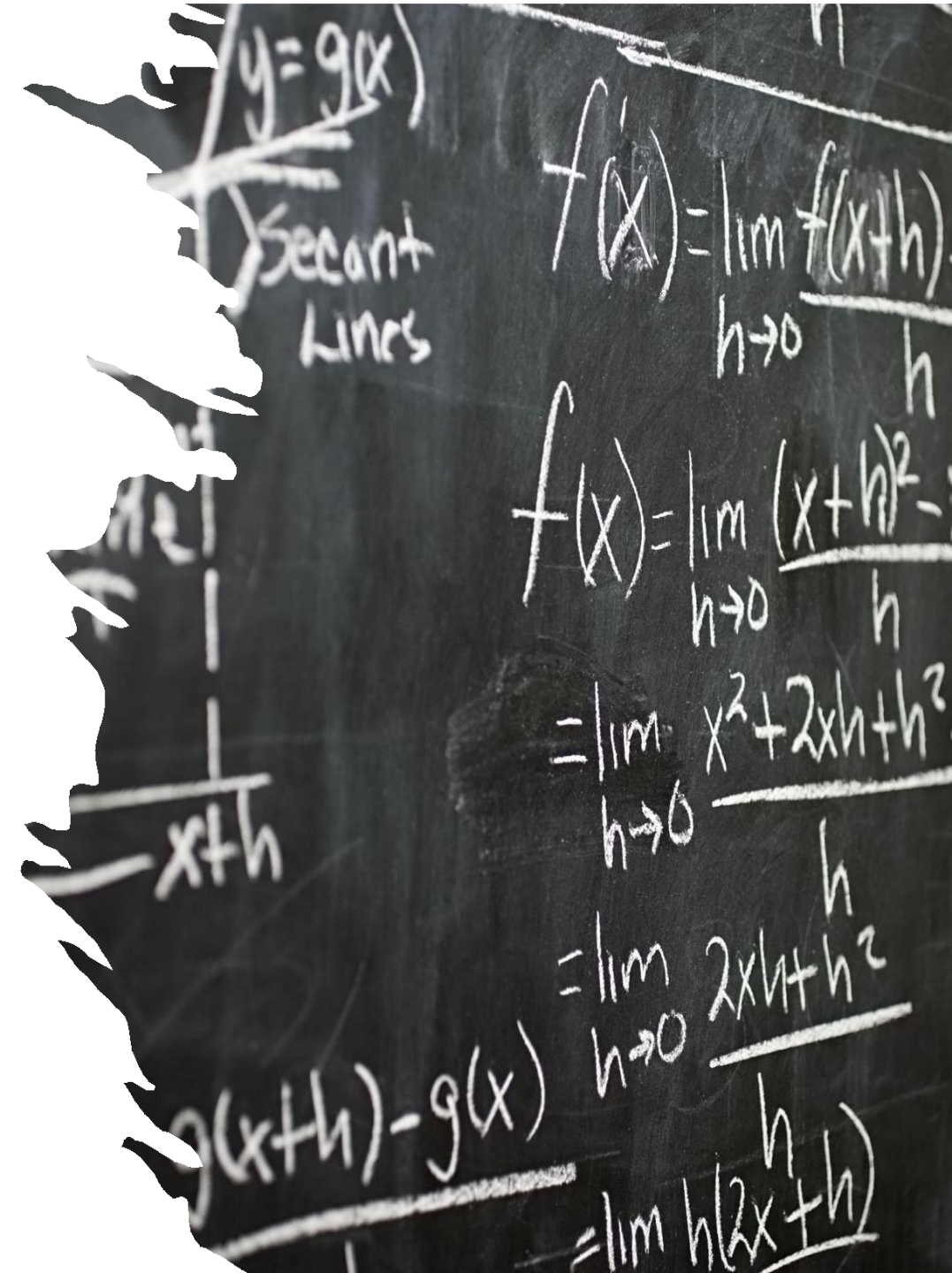
Değişken Tanımlama Kuralları:

İçeriği:

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

Doğru: degisken1, yas_23

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



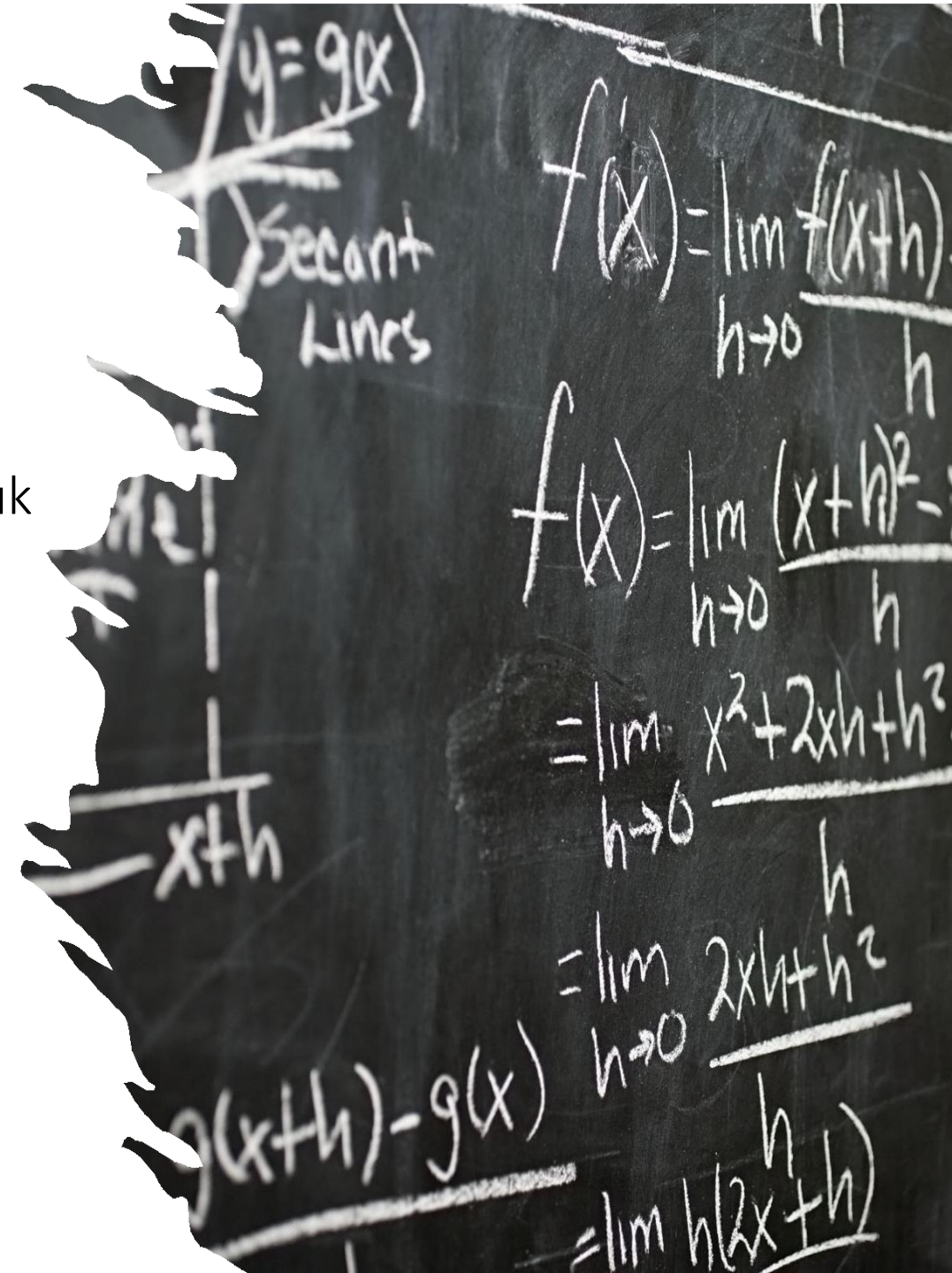
Değişken Tanımlama Kuralları:

Boşluklar Yok:

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

Doğru: ogrenci_adi, OgrenciAdi

Yanlış: ogrenci adi

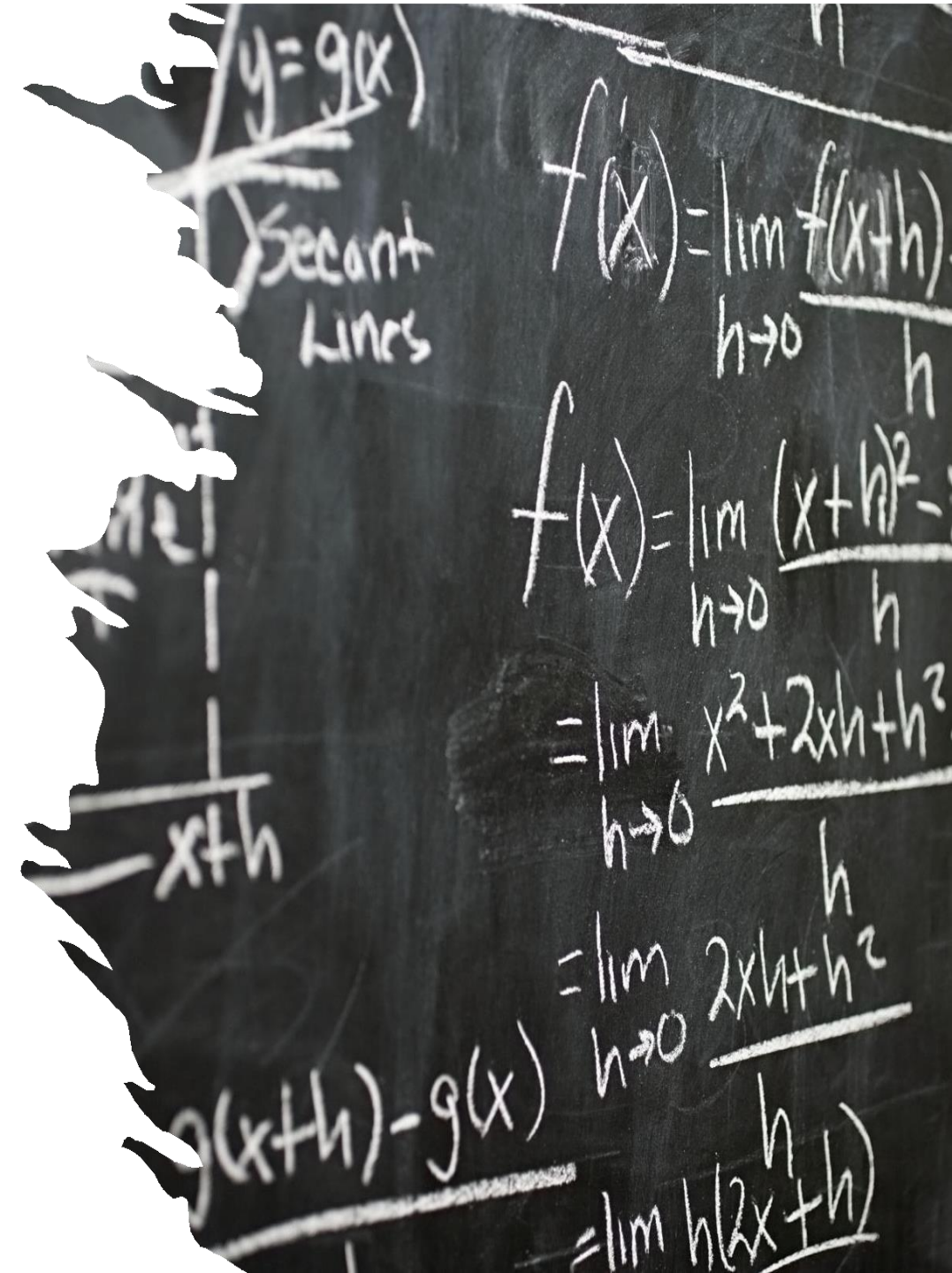


Değişken Tanımlama Kuralları:

Python Anahtar Kelimeleri Kullanılamaz:

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

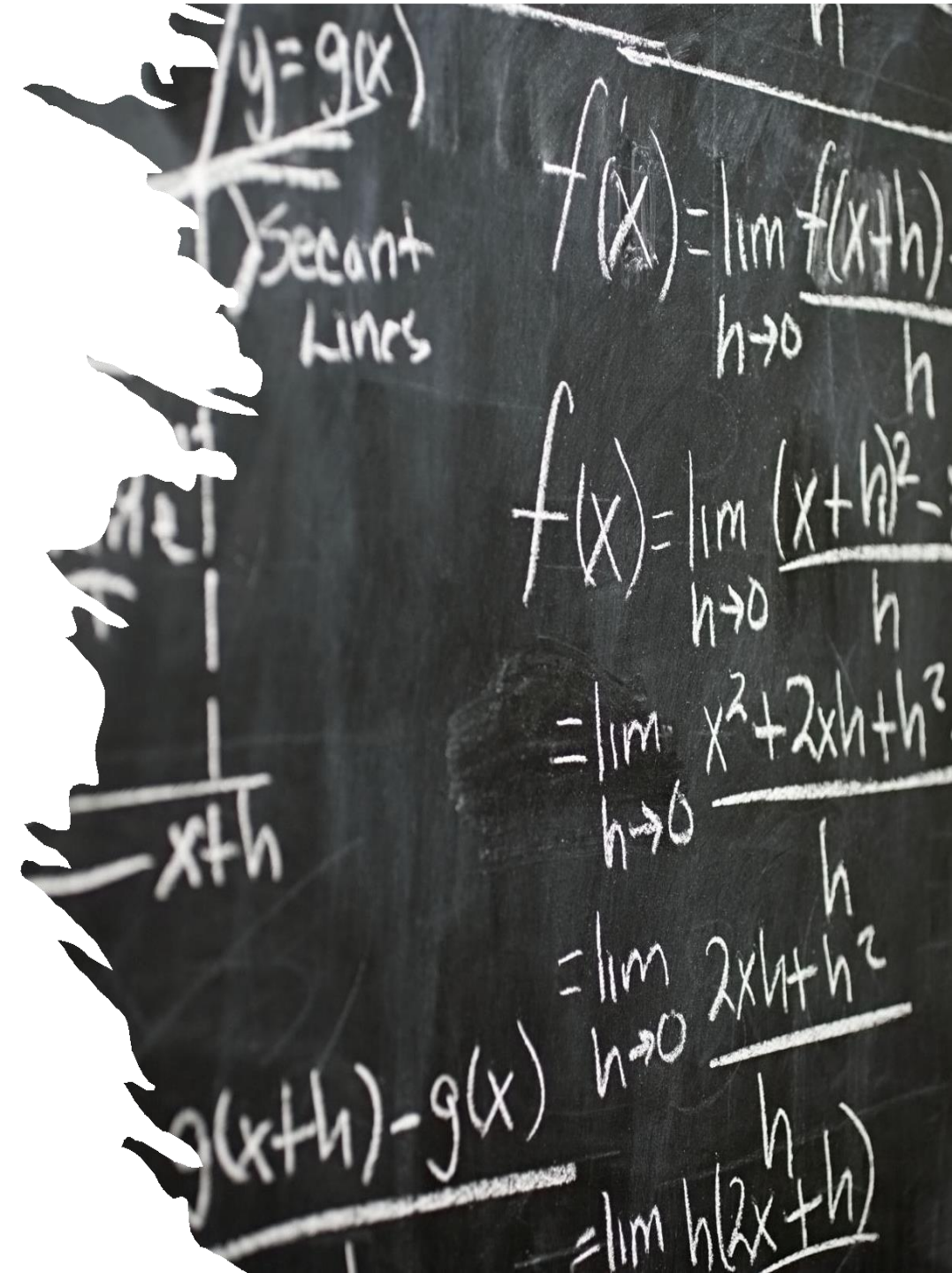
Yanlış: if = 10, True = 5



Değişken Tanımlama Kuralları:

Büyük-Küçük Harf Duyarlılığı:

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



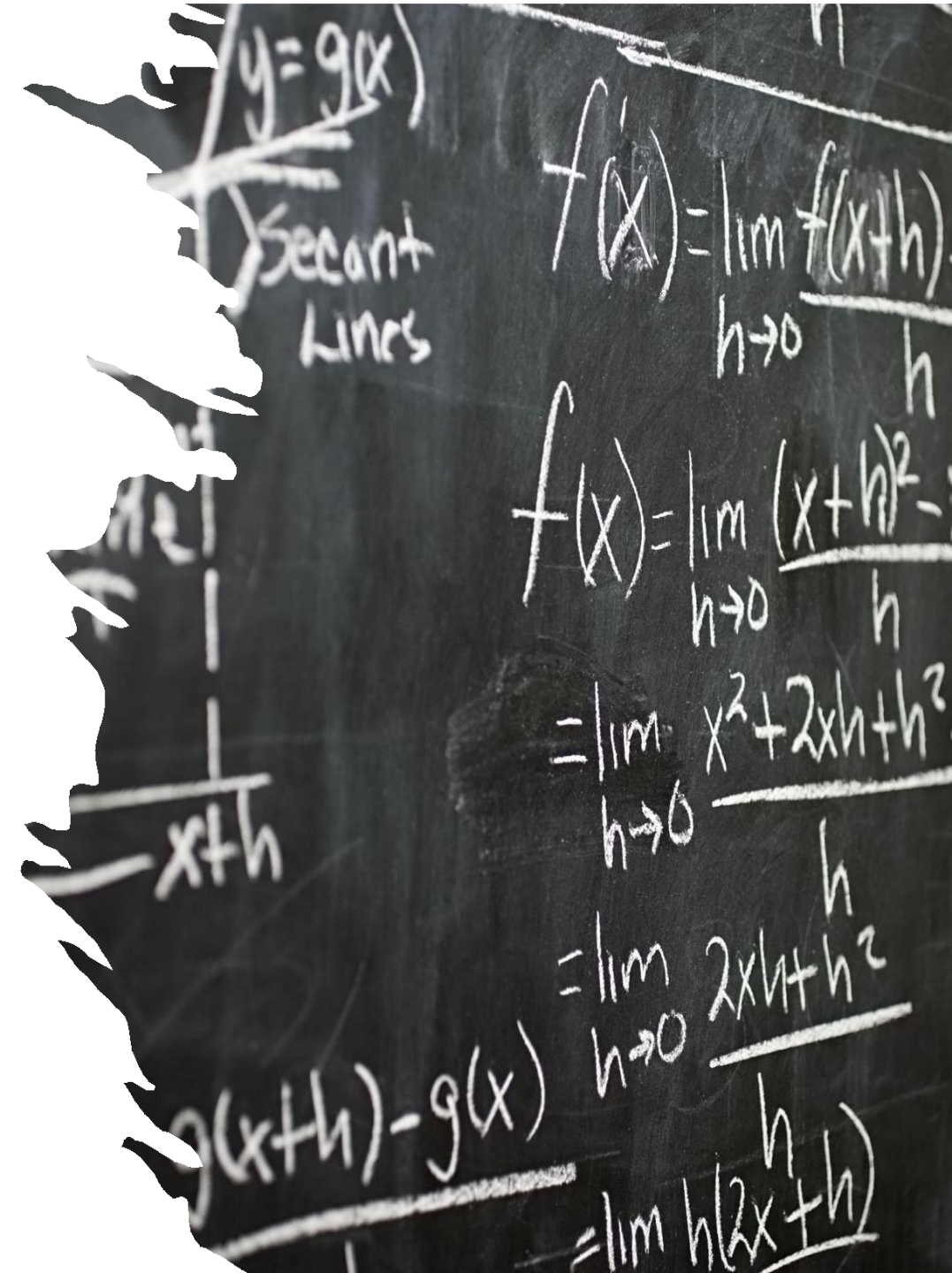
Değişken Tanımlama Kuralları:

Açıklayıcı İsimler Kullanma:

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

Doğru: `ogrenci_sayisi`, `ortalama_not`

Yanlış: `x`, `abc`



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 deęişkeni tanımlarken hangi tür veriyi tutacağıının belirlenmesi



Deęişkenler 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]
```


1.5. Tanımlanan deęişkenlerde işlem öncelięi

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	<code>3 + 2</code>	5
-	Çıkarma	<code>5 - 2</code>	3
*	Çarpma	<code>4 * 3</code>	12
/	Bölme	<code>10 / 2</code>	5.0
%	Mod (kalan bulma)	<code>10 % 3</code>	1
**	Üs alma	<code>2 ** 3</code>	8
//	Tamsayı bölmesi (floordiv)	<code>10 // 3</code>	3

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
<code>and</code>	Tüm koşullar doğruysa True	<code>True and False</code>	<code>False</code>
<code>or</code>	Koşullardan biri doğruysa True	<code>True or False</code>	<code>True</code>
<code>not</code>	Doğruyu yanlış, yanlış doğruya çevirir	<code>not True</code>	<code>False</code>

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	<code>x = 5</code>	<code>x = 5</code>
+=	Değeri toplar ve sonucu atar	<code>x += 3</code> (<code>x = x + 3</code>)	<code>x = 8</code>
-=	Değeri çıkarır ve sonucu atar	<code>x -= 2</code> (<code>x = x - 2</code>)	<code>x = 6</code>
*=	Değeri çarpar ve sonucu atar	<code>x *= 4</code> (<code>x = x * 4</code>)	<code>x = 24</code>
/=	Değeri böler ve sonucu atar	<code>x /= 2</code> (<code>x = x / 2</code>)	<code>x = 12.0</code>
%=	Mod alır ve sonucu atar	<code>x %= 5</code> (<code>x = x % 5</code>)	<code>x = 2.0</code>
**=	Üssünü alır ve sonucu atar	<code>x **= 3</code> (<code>x = x ** 3</code>)	<code>x = 8</code>
//=	Tamsayı bölümünü alır ve sonucu atar	<code>x //= 3</code> (<code>x = x // 3</code>)	<code>x = 2</code>

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)
```

BITTİ!

1. ÜNİTE SONU