Modül (kalan) işleci

→ bölümden kalan

→ yüzde (%) sembolü

```
>>> bolum = 7 / 3
>>> print bolum
2
>>> kalan = 7 % 3
>>> print kalan
1
```

Modül (kalan) işleci

→ iki sayının birbirine tam bölünüp bölünemediğini bu şekilde sınayabiliriz

```
1  def is_divisible(x, y):
2      return x % y == 0
```

Modül (kalan) işleci

→ sayının en sağ basamağındaki rakam veya rakamları elde edebiliriz

```
>>> x = 123
>>> x % 10
3
>>> x % 100
23
>>> _ % 10
3
```

boolean değerler ve deyimler

- → bool: yanlış/doğru
- → Boolean Cebrinin yaratıcısı George Boolen'dan
- → Bool cebri, modern bilgisayar aritmetiğinin temelidir
- → boolean değerler: True, False
- → BÜYÜK/küçük harf ayrımına dikkat

```
>>> type(True)
<type 'bool'>
>>> type(true)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

boolean deyimi

→ sonucu boolean değer olan deyimler

```
>>> x, y = 5, 6
>>> x == y  # esit mi?
False
>>> x == x
True
>>> x != y      # esit degil mi?
True
>>> x > y
False
>>> x < y
True
>>> x <= y
True
>>> x >= y
False
```

atama x karşılaştırma

→ tek eşittir “atama”

→ çift eşittir “karşılaştırma”

mantıksal işleçler

- → üç adettir: and, or, not
- → sırayla VE, VEYA, DEĞİL
- → Doğruluk tablosu, Logic Gates, De Morgan, Venn diagramları

```
>>> x, y = 5, 6
>>> x > 0
True
>>> x < 10
True
>>> x > 0 and x < 10
True
>>>
>>> x % 2 == 0 or x % 3 == 0
False
>>> x > y
False
>>> not(x > y)
True
```

koşullu yürütme

→ koşul cümleleri: ortalama notu 60'dan küçükse kaldı. Ama nasıl?

```
1  if ort < 60:
2      print "KALDI"
```

→ koşul cümleleri if ile kurulur

→ satırın sonundaki (işlevdekine benzer) iki noktaya dikkat!

→ if'den sonra gelen boolean deyime **koşul** denilir: ort $<$ 60

→ bu satırdan sonraki, girintili yazılanlar **gövde**

→ koşul doğruysa gövdedeki emirler yerine getirilir

→ koşul yanlışsa bir şey yapılmaz

if koşul yapısı

→ genel olarak

```
1  if KOSUL:
2      CUMLELER
```

→ if anahtar kelimesiyle başla

→ KOSUL ile devam et

→ bu ikisi başlık satırıdır

→ başlık satırını iki nokta -: ile bitir

→ takip eden girintili cümlelere **blok** adı verilir

→ ilk girintisiz cümle blok sonunu gösterir

→ bileşik cümlelerdeki cümle bloğuna cümlenin **gövdesi** denilir

→ gövdedeki cümleler koşul doğruysa yerine getirilir

→ if bileşik cümlesi en azından bir cümle içermelidir: pass kullanılabilir (boş cümle)

girintileme

→ C dilinde

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int ort = 65;
6      if (a < 60)
7      {
8      printf("Maalesef kaldiniz!\n");
9      return 0;
10     }
11 }
```

girintileme

→ C dilinde

```c
#include <stdio.h>

int main()
{
    int ort = 65;
    if (a < 60)
    {
    printf("Maalesef kaldiniz!\n");
    return 0;
    }
}
```

→ karmaşıkta olsa geçerli

```c
#include <stdio.h>
int main(){int a = 1;if (a == 1){printf("Merhaba Zalim Dünya!\
```

girintileme

→ Python'da yapmaya çalışırsak

```
1  ort = 65
2  if ort < 60:
3      print "Maalesef kaldiniz!"
```

girintileme

→ Python'da yapmaya çalışırsak

```
1  ort = 65
2  if ort < 60:
3      print "Maalesef kaldiniz!"
```

→ böyle yazamazsınız, hata alırsınız

```
1  ort = 65
2  if ort < 60:
3  print "Maalesef kaldiniz!"
```

→ evet python girintileme temelli bloklama-gövde yapısı kullanır

alternatif yürütme

→ koşul yanlışsa ne yapalım

```
1  if ortalama < 60:
2      print "KALDI"
3  else:
4      print "GECTI"
```

alternatif yürütme

→ koşul yanlışsa ne yapalım

```
1  if ortalama < 60:
2      print "KALDI"
3  else:
4      print "GECTI"
```

→ tek mi çift mi?

```
1  if x%2 == 0:
2      print x, "cifttir"
3  else:
4      print x, "tektir"
```

→ bu alternatiflere **dal** denilir

→ yürütme akışı farklı dallarla yoluna devam etmektedir

işlev

→ tek çift kontrolünü parity testinde kullanalım

```
1  def print_parity(x):
2      if x%2 == 0:
3          print x, "cifttir"
4      else:
5          print x, "tektir"
```

→ test edelim

```
1  >>> print_parity(17)
2  17 tektir
3  >>> y = 41
4  >>> print_parity(y+1)
5  42 cifttir
```

parola kontrol

→ girilen parolayı sına

```
1  parola = "python"
2  girdi  = raw_input("Lutfen parolanizi giriniz: ")
3
4  if girdi == parola:
5      print "Parola onaylandi!"
```

zincirleme koşul ifadeleri

→ bazen ikiden fazla durum olabilir: koşul doğru x değil

→ iki daldan fazlasına ihtiyaç duyarsak

→ zincirleme koşul ifadelerine başvururuz

```
if x < y:
    print "%s < %s" % (x, y)
elif x > y:
    print "%s > %s" % (x, y)
else:
    print "%s = %s" % (x, y)
```

→ sıcaklık - hal bilgisi

```
def hal(t):
    """\
        t sicaklik degerine (santigrat)

        >>> hal(-5)
        SIVI
        >>> hal(5)
        KATI
        >>> hal(105)
        GAZ
    """

    if t < 0:
        print "SIVI"
    elif t > 0 and t < 100:
        print "KATI"
    else:
        print "GAZ"
```

gerçeksi bir uygulama

→ demo: d04_4islem.py dosyasını iyileştirelim

içiçe koşul deyimleri

→ koşul deyimleri içiçe olabilir

```
1  if x > 0:
2      print "Kuzey-",
3
4      if y > 0:
5          print "Dogu"
6      else:
7          print "Bati"
8  else:
9      print "Guney-",
10
11     if y > 0:
12         print "Dogu"
13     else:
14         print "Bati"
```

içiçe koşul deyimleri

→ aşağıdaki kodu tek bir cümleyle yazabiliriz

```
1  if 0 < x:
2      if x < 10:
3      print "x pozitif ve tek basamaklıdır."
```

→ nasıl?

içiçe koşul deyimleri

→ aşağıdaki kodu tek bir cümleyle yazabiliriz

```
1  if 0 < x:
2      if x < 10:
3      print "x pozitif ve tek basamaklıdır."
```

→ nasıl?

```
1  if 0 < x and x < 10:
2          print "x pozitif ve tek basamaklıdır."
```

→ alternatif olarak

```
1  if 0 < x < 10:
2      print "x pozitif ve tek basamaklıdır."
```

geri dönüş cümlesi

→ return anahtar kelimesi işlevin o andan sonrasını bırak, çık

```
1  def print_square_root(x):
2      if x <= 0:
3      print "Sadece pozitif sayılar lütfen."
4      return
5
6      result = x**0.5
7      print "x'in kare kökü", result
```

→ erken çıkma

→ uygun hata mesajı

→ return ile ilgili daha fazlası sonradan gelecek

klavye girdisi

→ raw_input, input

→ raw_input, dizgi döndürür

→ input, eval(raw_input(prompt)) yani raw_input'un hesaplanmış/değerlendirilmiş hali

```
>>> val = raw_input("aritmetik ifade girin: ")
aritmetik ifade girin: 3 + 4
>>> val
'3 + 4'
>>> val = input("aritmetik ifade girin: ")
aritmetik ifade girin: 3 + 4
>>> val
7
```

→ raw_input, 3 + 4'ü dizgi olarak aldı ve çıktı üretti: '3 + 4'

→ input, 3 + 4 ifadesini hesapladı, sonucu aldı ve çıktı üretti: 7

klavye girdisi

→ input ile dizgi girerken dikkatli olun

→ kabukta dizgi girme kuralları geçerli

```
>>> str = raw_input("dizgi girin: ")
dizgi girin: python programlama dili
>>> str
'python programlama dili'
>>> str = input("dizgi girin: ")
dizgi girin: python programlama dili
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1
    python programlama dili
               ^
SyntaxError: invalid syntax
>>> str = input("dizgi girin: ")
dizgi girin: "python programlama dili"
>>> str
'python programlama dili'
```

kompozisyon

→ işlev, girdi, kontrol, in anahtar kelimesi

```
1  def ask_ok(prompt):
2      ok = raw_input(prompt)
3      if ok in ('y', 'ye', 'yes'):
4          print 'YES'
5      if ok in ('n', 'no', 'nop', 'nope'):
6          print 'NO'
```

→ prompt biçiminde değişken kullanabiliriz

→ koşul yapısı içerisinde in'i kullanabiliriz

→ in, değer ardışıllığın içerisinde var mı?

→ yukarıdaki işlevi çağırırken

```
1  ask_ok('Cikmak istediginizden emin misiniz?')
```

dizgiye dönüşüm

→ dizgiye dönüşüm: str(ARGUMAN)

```
>>> str(32)
'32'
>>> str(3.14)
'3.14'
>>> str(True)
'True'
```

→ 32 farklıdır '32'

→ dizgi mi? sayı mı?

tip dönüşümleri

→ tamsayıya dönüşüm: int(ARGUMAN)

→ dönüştürmeye çalış, başarısız olursan hata mesajı ver

```
>>> int("32")
32
>>> int("32", 16)
50
>>> int(str(32), 8)
26
>>> int("merhaba")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'merhaba'
>>> val = raw_input("Bir deger girin: ")
Bir deger girin: 34
>>> int(val)
34
>>> int("0x13", 16)
19
```

→ raw_input, dizgi döndürür

→ dizgi $\Longrightarrow$ *say*ß*dönüşümüiçin* : int()

→ gerçel $\Longrightarrow$ *tamsay*ß*d*ö*n*üş*m*ü

```
>>> int(34.4)
34
>>> int(34.999)
34
>>> int("34")
34
>>> int("34.4")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '34.4'
```

→ kural: tamsayı kısmını al

→ dönüşümler

```
>>> float(32)
32.0
>>> float("3.141519")
3.1415190000000002
>>> int("3.141519")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.141519'
```

boolean dönüşümü

→ mantıksala dönüşüm: bool()

```
>>> bool(1)
True
>>> bool(1.0)
True
>>> bool(0)
False
>>> bool(0.0)
False
>>> bool("merhaba")
True
>>> bool("")
False
```

→ kural: boş veya sıfır değerliler False'dur

kısaltmalar

→ ad için kullanıcı bir şeyle girmiş mi?

```
>>> ad = raw_input("Adinizi giriniz: ")
Adinizi giriniz:
>>> if ad == "":
...     print "adi girmeden devam edemezsiniz"
...
adi girmeden devam edemezsiniz
```

→ önce yardımcı bilgi: >>> bool(ad), eğer ad boşsa False üretir

→ buradaki if ad== “”: kısmını artık kısaltabiliriz

→ önce böyle >>> if bool(ad):

→ sonra şöyle >>> if ad:

hata ayıklama

→ bir hata meydana geldiğinde bir çok bilgi sunar

→ en faydalı olanları

1. hatanın türü
2. nerede olduğu

→ Syntax error, bulması kolaydır

```
>>> x = 5
>>>  y = 6
  File "<stdin>", line 1
y = 6
^
IndentationError: unexpected indent
```

→ genel olarak hatalar belirtilen kodun/satırın öncesindedir

çalışma zamanı hatası

→ amaç SNR'yi hesaplamak

```
1  # d04_debug.py
2
3  import math
4  signal_power = 9
5  noise_power  = 10
6  ratio = signal_power / noise_power
7  decibels = 10 * math.log10(ratio)
8  print decibels
```

→ çalıştırıldığında

```
1  $ python d04_debug.py
2  Traceback (most recent call last):
3    File "d04_debug.py", line 7, in <module>
4  decibels = 10 * math.log10(ratio)
5  ValueError: math domain error
```

→ satır7'de nedir acaba?

çözüm

d04_debug2.py dosyası

GASP

→ Gasp: Graphics API for Students of Python

→ Python için Grafik Uygulama Geliştirme Arayüzü

→ önce kur $ sudo apt-get install python-gasp

→ kodla

```
>>> from gasp import *
>>> begin_graphics()
>>> Circle((200, 200), 60)
Circle instance at (200, 200) with radius 60
>>> Line((100, 400), (580, 200))
Line instance from (100, 400) to (590,
>>> Box((400, 350), 120, 100)
Box instance at (400, 350) with width 120 and height 100
>>> end_graphics()
```

→ GASP'ı bilgisayar programlama kavramlarını görselleştirmek ve öğrenirken eğlenmek için kullanacağız.

sıra sizde

- → Fermat's Last Theorem: a^n + b^n = c^n
- → d04_fermat.py
- → d04_istriangle.py
- → d04_html.py
- → kur bilgisi