NUMBERS

Nomor tipe data menyimpan nilai numerik. Mereka adalah tipe data yang tidak berubah, artinya mengubah nilai dari sejumlah hasil tipe data pada objek yang baru dialokasikan. Nomor objek dibuat saat Anda memberikan nilai pada mereka. Contohnya –

var1 = 1

var2 = 10

Anda juga dapat menghapus referensi ke objek nomor dengan menggunakan del statement. Sintaks dari pernyataan del adalah -

del var1[,var2[,var3[....,varN]]]]

Anda dapat menghapus satu objek atau beberapa objek dengan menggunakan pernyataan del. Sebagai contoh:

del var

del var _a, var _b

Python mendukung empat jenis numerik yang berbeda –

• int (signed integers): Mereka sering disebut bilangan bulat atau int, bilangan bulat positif

atau negatif tanpa titik desimal.

• panjang (bilangan bulat panjang): Juga disebut rindu, bilangan bulat adalah ukuran tak terbatas, ditulis seperti bilangan bulat dan diikuti huruf besar atau huruf kecil L.

• float (floating point real value): Disebut juga floats, mereka mewakili bilangan real dan ditulis dengan titik desimal membagi bilangan bulat dan bagian fraksional. Mengapung juga bisa dalam notasi ilmiah, dengan E atau e menunjukkan kekuatan 10 (2,5e2 = 2,5 x 102 = 250).

• kompleks (bilangan kompleks): berbentuk a + bJ, di mana a dan b mengapung dan J (atau j) mewakili akar kuadrat -1 (yang merupakan bilangan imajiner). Bagian sebenarnya dari bilangan tersebut adalah a, dan bagian imajinernya adalah b. Nomor kompleks tidak banyak digunakan dalam pemrograman Python.

CONTOH

Berikut adalah beberapa contoh angka

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEL	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

• Python memungkinkan Anda menggunakan huruf kecil L dengan panjang, namun disarankan agar Anda hanya menggunakan huruf besar L untuk menghindari kebingungan dengan nomor 1. Python menampilkan bilangan bulat panjang dengan huruf besar L.
• Nomor kompleks terdiri dari sepasang bilangan floating point asli yang ditandai dengan tanda + bj, di mana a adalah bagian sebenarnya dan b adalah bagian imajiner dari bilangan kompleks.

Konversi Tipe Jumlah

Python mengubah nomor secara internal dalam sebuah ekspresi yang mengandung tipe campuran untuk tipe umum untuk evaluasi. Tapi terkadang, Anda perlu memaksa nomor secara eksplisit dari satu jenis ke tipe lain untuk memenuhi persyaratan parameter operator atau fungsi.

- Type **int(x)** to convert x to a plain integer.
- Type **long(x)** to convert x to a long integer.
- Type **float(x)** to convert x to a floating-point number.
- Type **complex(x)** to convert x to a complex number with real part x and imaginary part zero.
- Type **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Fungsi Matematika

Python mencakup fungsi berikut yang melakukan perhitungan matematis.

Function	Returns (description)
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: ex
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0 .
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
pow(x, y)	The value of x**y.
round(x [,n])	**x** rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0

Fungsi Nomor Acak

Nomor acak digunakan untuk aplikasi permainan, simulasi, pengujian,

keamanan, dan privasi. Python mencakup fungsi berikut yang umum digunakan.

Function	Description
choice(seq)	A random item from a list, tuple, or string.
randrange ([start,] stop [,step])	A randomly selected element from range(start, stop, step)
random()	A random float r, such that 0 is less than or equal to r and r is less than 1
seed([x])	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
shuffle(lst)	Randomizes the items of a list in place. Returns None.
uniform(x, y)	A random float r, such that x is less than or equal to r and r is less than y

FUNGSI TRIGONOMETRIK

Python mencakup fungsi berikut yang melakukan perhitungan trigonometri.

Function	Description
acos(x)	Return the arc cosine of x, in radians.
asin(x)	Return the arc sine of x, in radians.
atan(x)	Return the arc tangent of x, in radians.
atan2(y, x)	Return atan(y / x), in radians.
cos(x)	Return the cosine of x radians.
hypot(x, y)	Return the Euclidean norm, sqrt(x*x + y*y).
sin(x)	Return the sine of x radians.
tan(x)	Return the tangent of x radians.
degrees(x)	Converts angle x from radians to degrees.
radians(x)	Converts angle x from degrees to radians.

Konstanta matematika

Modul ini juga mendefinisikan dua konstanta matematika –

Constants	Description
pi	The mathematical constant pi.
e	The mathematical constant e.

String adalah salah satu jenis yang paling populer dengan Python. Kita bisa membuatnya hanya dengan melampirkan karakter dalam tanda kutip. Python memperlakukan tanda petik tunggal sama dengan tanda kutip ganda. Membuat string semudah memberi nilai pada sebuah variabel. Misalnya –
var1 = ’Hello World!’

var2 = "Python Programming"

Mengakses Nilai dalam String
Python tidak mendukung tipe karakter; Ini diperlakukan sebagai string dengan panjang satu,

sehingga juga dianggap sebagai substring.
Untuk mengakses substring, gunakan tanda kurung siku untuk mengiris beserta indeks atau indeks untuk mendapatkan substring Anda. Misalnya -

```
#!/usr/bin/python

var1 = ’Hello World!’
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
var1[0]:  H
var2[1:5]:  ytho
```

Memperbarui String

Anda dapat "memperbarui" string yang ada dengan (kembali) menugaskan variabel ke string lain. Nilai baru dapat dikaitkan dengan nilai sebelumnya atau ke string yang sama sekali berbeda sama sekali. Misalnya -

```
#!/usr/bin/python

var1 = ’Hello World!’

print "Updated String :- ", var1[:6] + ’Python’
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Updated String :- Hello Python
```

Karakter melarikan diri

Tabel berikut adalah daftar karakter escape atau non-printable yang dapat diwakili dengan notasi backslash.
Karakter pelarian ditafsirkan; dalam satu dikutip serta dua kali mengutip string.

Backslashnotation	Hexadecimalcharacter	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M- \C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

String Operator Khusus

Asumsikan variabel string memegang ’Halo’ dan variabel b berisi ’Python’, lalu –

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r' \n' prints \n and print R' \n'prints \n
%	Format - Performs String formatting	See at next section

Penyandian String Operator
Salah satu fitur Python yang paling keren adalah format string operator %. Operator ini unik untuk string dan membuat paket memiliki fungsi dari keluarga printf C () C. Berikut adalah contoh sederhana -

```
#! / Usr / bin / python
 Cetak "Nama saya % s dan beratnya adalah % d kg!" % ('Zara', 21)
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
Nama saya Zara dan beratnya adalah 21 kg!
```

Berikut adalah daftar lengkap simbol yang bisa digunakan bersamaan dengan % -

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase ’e’)
%E	exponential notation (with UPPERcase ’E’)
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Simbol dan fungsionalitas pendukung lainnya tercantum dalam tabel berikut –

Symbol	Functionality
*	argument specifies width or precision
-	left justification
+	display the sign
<sp>	leave a blank space before a positive number
#	add the octal leading zero (’0’) or hexadecimal leading ’0x’ or ’0X’, depending on whether ’x’ or ’X’ were used.
0	pad from left with zeros (instead of spaces)
%	’ % %’ leaves you with a single literal ’ %’
(var)	mapping variable (dictionary arguments)
m.n.	m is the minimum total width and n is the number of digits to display after the decimal point (if appl.)

Triple Quotes

Tiga tanda kutip Python hadir untuk menyelamatkannya dengan membiarkan string meman-

jang banyak baris, termasuk kata kunci NEWLINEs, TABs, dan karakter khusus lainnya.
Sintaks untuk triple quotes terdiri dari tiga tanda kutip tunggal atau ganda berturut-turut.
#! / Usr / bin / python
para _str = "" "ini adalah string panjang yang terdiri dari

Beberapa baris dan karakter yang tidak dapat dicetak seperti
TAB (\ t) dan mereka akan muncul seperti itu saat ditampilkan.

NEWLINEs dalam string, apakah secara eksplisit diberikan seperti
Ini dalam tanda kurung [\ n], atau hanya NEWLINE di dalamnya
tugas variabel juga akan muncul.
"" "
Cetak para _str
Bila kode diatas dieksekusi, maka hasilnya akan menghasilkan hasil berikut. Perhatikan bagaimana setiap karakter khusus telah diubah menjadi bentuk cetaknya, sampai ke NEW-LINE terakhir di akhir string antara "up". Dan menutup tanda kutip tiga kali. Perhatikan juga bahwa NEWLINEs terjadi baik dengan carriage return yang eksplisit di akhir baris atau kode escape-nya (\ n) -
Ini adalah string panjang yang terdiri dari
beberapa baris dan karakter yang tidak dapat dicetak seperti
TAB () dan mereka akan muncul seperti itu saat ditampilkan.
NEWLINEs dalam string, apakah secara eksplisit diberikan seperti
ini dalam tanda kurung [
], atau hanya NEWLINE di dalamnya
tugas variabel juga akan muncul.
String mentah tidak memperlakukan garis miring terbalik sebagai karakter spesial sama sekali. Setiap karakter yang Anda masukkan ke dalam string mentah tetap seperti yang Anda tulis -
#! / Usr / bin / python
cetak 'C: \ \ tempat'

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -
C: di mana-mana
Sekarang mari kita gunakan string mentah. Kami akan mengutarakan ekspresi 'sebagai berikut -
#! / Usr / bin / python
Cetak r'C: \ \ tempat '

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -
C: tidak di mana-mana
String Unicode
String normal dengan Python disimpan secara internal sebagai 8-bit ASCII, sedangkan string Unicode disimpan sebagai Unicode 16-bit. Hal ini memungkinkan untuk serangkaian karakter yang lebih bervariasi, termasuk karakter khusus dari kebanyakan bahasa di dunia. Saya akan membatasi perlakuan saya terhadap string Unicode sebagai berikut -
#! / Usr / bin / python
cetak u'Hello, dunia! '

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -
Halo Dunia!
Seperti yang bisa Anda lihat, senar Unicode menggunakan awalan Anda, sama seperti senar biasa menggunakan awalan r.
Metode String Terpadu
Python menyertakan metode built-in berikut untuk memanipulasi string –

SN	Methods with Description
1	capitalize() Capitalizes first letter of string
2	center(width, fillchar) Returns a space-padded string with the original string centered to a total of width columns.
3	count(str, beg= 0,end=len(string)) Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
4	decode(encoding='UTF-8',errors='strict') Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
5	encode(encoding='UTF-8',errors='strict') Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
6	endswith(suffix, beg=0, end=len(string)) Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
7	expandtabs(tabsize=8) Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
8	find(str, beg=0 end=len(string)) Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
9	index(str, beg=0, end=len(string)) Same as find(), but raises an exception if str not found.
10	isalnum() Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
11	isalpha() Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
12	isdigit() Returns true if string contains only digits and false otherwise.
13	islower() Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
14	isnumeric() Returns true if a unicode string contains only numeric characters and false otherwise.
15	isspace() Returns true if string contains only whitespace characters and false otherwise.
16	istitle() Returns true if string is properly "titlecased" and false otherwise.
17	isupper() Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
18	join(seq) Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
19	len(string) Returns the length of the string
20	ljust(width[, fillchar]) Returns a space-padded string with the original string left-justified to a total of width columns.
21	lower() Converts all uppercase letters in string to lowercase.
22	lstrip() Removes all leading whitespace in string.
23	maketrans() Returns a translation table to be used in translate function.
24	max(str) Returns the max alphabetical character from the string str.
25	min(str) Returns the min alphabetical character from the string str.
26	replace(old, new [, max]) Replaces all occurrences of old in string with new or at most max occurrences if max given.
27	rfind(str, beg=0,end=len(string)) Same as find(), but search backwards in string.
28	rindex(str, beg=0, end=len(string)) Same as index(), but search backwards in string.
29	rjust(width,[, fillchar]) Returns a space-padded string with the original string right-justified to a total of width columns.
30	rstrip() Removes all trailing whitespace of string.
31	split(str="", num=string.count(str)) Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
32	splitlines(num=string.count('\n')) Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
33	startswith(str, beg=0,end=len(string)) Determines if string or a substring of string (if start-

9.1.2.2. Implementing the arithmetic operations

We want to implement the arithmetic operations so that mixed-mode operations either call an implementation whose author knew about the types of both arguments, or convert both to the nearest built in type and do the operation there. For subtypes of Integral , this means that _ _add _ _() and _ _radd _ _() should be defined as:

```
class MyIntegral(Integral):
    def _ _add _ _(self, other):

      if isinstance(other, MyIntegral):
         return do _my _adding _stuff(self, other)
      elif isinstance(other, OtherTypeIKnowAbout):
         return do _my _other _adding _stuff(self, other)
      else:
         return NotImplemented
    def _ _radd _ _(self, other):

      if isinstance(other, MyIntegral):
         return do _my _adding _stuff(other, self)
      elif isinstance(other, OtherTypeIKnowAbout):
         return do _my _other _adding _stuff(other, self)
      elif isinstance(other, Integral):
         return int(other) + int(self)
      elif isinstance(other, Real):
         return float(other) + float(self)
      elif isinstance(other, Complex):
         return complex(other) + complex(self)
      else:
         return NotImplemented
```

There are 5 different cases for a mixed-type operation on subclasses of Complex . I'll refer to all of the above code that doesn't refer to MyIntegral and OtherTypeIKnowAbout as "boilerplate ". a will be an instance of A, which is a subtype of Complex (a : A <: Complex), and b : B <: Complex. I'll consider a + b:

1. If A defines an _ _add _ _() which accepts b, all is well.
2. If A falls back to the boilerplate code, and it were to return a value from _ _add _ _() , we'd miss the possibility that B defines a more intelligent _ _radd _ _() , so the boilerplate should return NotImplemented from _ _add _ _() . (Or A may not implement _ _add _ _() at all.)
3. Then B's _ _radd _ _() gets a chance. If it accepts a, all is well.
4. If it falls back to the boilerplate, there are no more possible methods to try, so this is where the default implementation should live.
5. If B <: A, Python tries B. _ _radd _ _ before A. _ _add _ _. This is ok, because it was implemented with knowledge of A, so it can handle those instances before delegating to Complex .

If A <: Complex and B <: Real without sharing any other knowledge, then the appropriate shared operation is the one involving the built in complex , and both _ _radd _ _() s land there, so a+b == b+a.
Because most of the operations on any given type will be very similar, it can be useful to define a helper function which generates the forward and reverse instances of any given operator. For example, fractions.Fraction uses:

```
def _operator _fallbacks(monomorphic _operator, fallback _operator):
    def forward(a, b):
        if isinstance(b, (int, long, Fraction)):
            return monomorphic _operator(a, b)
        elif isinstance(b, float):
            return fallback _operator(float(a), b)
        elif isinstance(b, complex):
            return fallback _operator(complex(a), b)
        else:
            return NotImplemented
    forward. _ _name _ _ = ' _ _' + fallback _operator. _ _name _ _ + ' _ _'
    forward. _ _doc _ _ = monomorphic _operator. _ _doc _ _
     def reverse(b, a):

        if isinstance(a, Rational):
            # Includes ints.
            return monomorphic _operator(a, b)
        elif isinstance(a, numbers.Real):
            return fallback _operator(float(a), float(b))
        elif isinstance(a, numbers.Complex):
            return fallback _operator(complex(a), complex(b))
        else:
            return NotImplemented
    reverse. _ _name _ _ = ' _ _r' + fallback _operator. _ _name _ _ + ' _ _'
    reverse. _ _doc _ _ = monomorphic _operator. _ _doc _ _
     return forward, reverse

def _add(a, b):

    """a + b"""
    return Fraction(a.numerator * b.denominator +
                    b.numerator * a.denominator,
                    a.denominator * b.denominator)
_ _add _ _, _ _radd _ _ = _operator _fallbacks( _add, operator.add)

# ...
```