PYTHON

INTRODUCTION

Variables and assignments

★ Integer

★ Float

★ String

★ List

★ Arrays

Integers

binary representation

Integers can be of any length

Operators	Description
+, -, *, /	Arithmatic
%	modulus
x**a	power
I, ^, &	Bitwise or, exclusive or, and,
~x	bitwise complement
<, >, == , <=, >=,	relational operator

```
In [1]: x=2**31-1

In [2]: x
Out[2]: 2147483647
```

Binary equivalent in Hex, bin, or oct format

```
In [3]: hex(x)
Out[3]: '0x7fffffff'
```

```
In [4]: 5%3
Out[4]: 2

In [5]: 5/3
Out[5]: 1

In [6]:
100&110
Out[6]: 100

In [7]: 100 |
110
Out[7]: 110
```

Real numbers

Fixed point representation

$$f = \text{sign} \times (a_n 10^n + .. + a_0 + a_{-1} 10^{-1} + ... + a_{-m} 10^{-m})$$

In decimal with n=1, m=2

00.00, 00.01, ..55.54, ., .99.99

difference: 0.01

Smallest: 0.01; largest: 99.99

uniformly spaced

Floating point representation

$$f = (a_0 + a_{-1}10^{-1} + \ldots + a_{-m}10^{-m}) \times 10^{E}$$

with $a_0 > 0$

m+1 significant digits, some digits for *E*

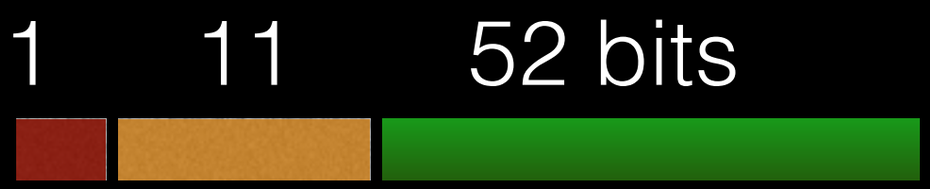
Example: m=2 and 2 digits for *E*

range: 1.0×10^{-99} to 9.9×10^{99}

Uniform for one *E*, then there is a jump

1.0,1.1,1.2, ...,2.1, ..., 9.9,

1.0x10, 1.1x10, ..., 2.1x10, ..., 9.9x10



$$f = (-1)^{\text{sign}} (1.b_{-1} b_{-2} \ldots b_{-52}) \times 2^{e-1023}$$

$$f = (-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{-i} 2^{-i}\right) \times 2^{e-1023}$$

Exponent: -1023 log2 ≈ -308 to
1025 log 2 ≈ 308

Precision: $2^{-52} \approx 2.22 \times 10^{-16}$

Decimal to bin

$(1.1)_{10}$=?

0.1→0.2→0.4→0.8→1.6 →0.6→1.2
→0.2→0.4→0.8→1.6→0.6→1.2→0.2

$(1.1)_{10} = (1.00011001...)_2 = (1.00011001)_2 =$
'0x1.199999999999ap+0'

```
In [8]: x=1.1

In [9]: float.hex(x)
Out[9]: '0x1.199999999999ap+0'

In [10]: x=1.5

In [11]: float.hex(x)
Out[11]: '0x1.8000000000000p+0'
```

Complex Float

```
In [12]: x=1+1j

In [13]: x*x, x*2, abs(x), conj(x)
Out[13]: (2j, (2+2j), 1.4142135623730951, (1-1j))
```

Strings

```
In [14]: x = "This is a string.”

In [15]: x
Out[15]: 'This is a string.’
```

```
In [17]: x+"  Hi, string?"
Out[17]: 'This is a string.  Hi, string?'
```

Input/Output

```
In [18]: x=5

In [19]: print "x=", x
x= 5
```

```
In [20]: x = input("Enter the value of x =")
Enter the value of x =10
```

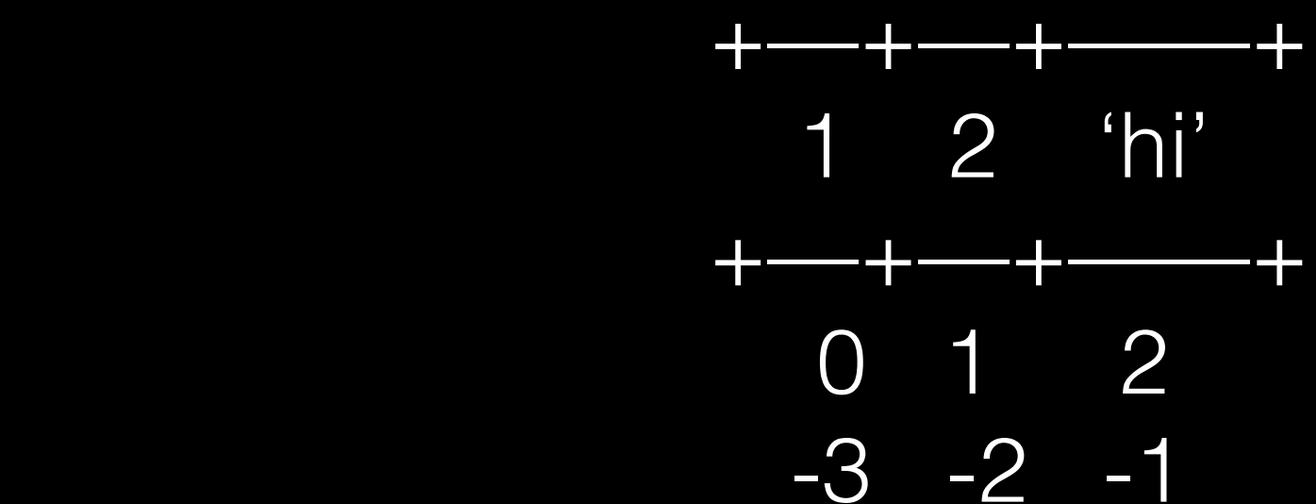
```
In [21]: x
Out[21]: 10
```

List

List of quantities: could be of different types

```
In [22]: y = [1,2,'hi']
```

Access elements using y[i]

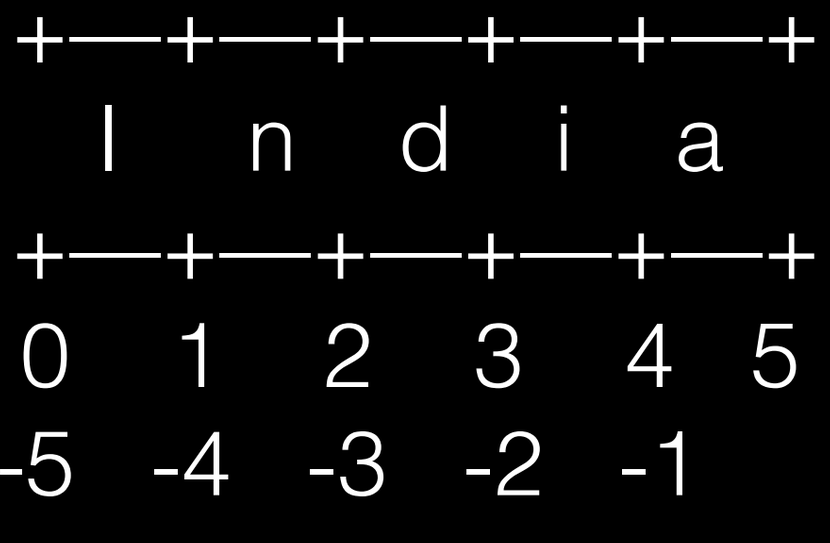



```
In [26]: print y[0], y[1], y[2], y[-1], y[-2]
1 2 hi hi 2
```

```
In [27]: del y[1]

In [28]: y

Out[28]: [1, 'hi']
```

```
In [22]: y = [1,2,'hi']

In [23]: y*2
Out[23]: [1, 2, 'hi', 1, 2, 'hi']

In [24]: y+[10]
Out[24]: [1, 2, 'hi', 10]

In [25]: y.append(3)
In [27]: y.append('computers')

In [28]: y
Out[28]: [1, 2, 'hi', 3, 'computers']

In [36]: y.pop(2)
Out[36]: 'hi'
```

Slicing




```
In [19]: x="india"

In [20]: x[0:3]
Out[20]: 'ind'

In [21]: range(0,3)
Out[21]: [0, 1, 2]
```

Numpy Arrays

Numerical Python

Arrays are more efficient than lists
because the data size is fixed for arrays.

The datatype can of int, float, string.

We focus on float arrays in this course.

```
In [35]: y=array([1,2])

In [36]: y
Out[36]: array([1, 2])

In [38]: dtype(y[0]), dtype(y[1])
Out[38]: (dtype('int64'), dtype('int64'))

In [39]: y*2
Out[39]: array([2, 4])
```

```
In [45]: size(y)
Out[45]: 2

In [46]: len(y)
Out[46]: 2

In [48]: z = y+3

In [49]: z
Out[49]: array([4, 5])
```

For mixed inputs

```
In [33]: y=array([1.0,2])

In [34]: dtype(y[0]), dtype(y[1])
Out[34]: (dtype('float64'), dtype('float64'))


In [35]: y=array([1.0,2.0])

In [36]: dtype(y[0]), dtype(y[1])
Out[36]: (dtype('float64'), dtype('float64'))
```

```
In [38]: x=array([[0,1],[1,0]])

In [39]: x
Out[39]:  array([[0, 1], [1, 0]])


In [40]: det(x)
Out[40]: -1.0

In [41]: eig(x)
Out[41]: (array([ 1., -1.]), array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]]))
```

Accessing array elements

[0,0]	[0,1]
[1,0]	[1,1]

```
In [19]: print x[0,1], x[0][1]
1 1
```

```
In [40]: y=eig(x)

In [41]: y
Out[41]:
(array([ 1., -1.]), array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]]))


In [42]: y[0,0]
TypeError: tuple indices must be integers, not tuple


In [43]: print y[1][1], y[1][1][1]
[ 0.70710678  0.70710678] 0.707106781187
```

```
In [53]: y
Out[53]: array([1, 2])

In [54]: z
Out[54]: array([4, 5])

In [55]: sqrt(y)
Out[55]: array([ 1. ,  1.41421356])
```

```
In [56]: y+z
Out[56]: array([5, 7])

In [57]: y*z
Out[57]: array([ 4, 10])
```

```
In [22]: arange(0,3)
Out[22]: array([0, 1, 2])

In [23]: zeros(4)
Out[23]: array([ 0.,  0.,  0.,  0.])

In [24]: ones(4)
Out[24]: array([ 1.,  1.,  1.,  1.])

In [25]: empty(4)
Out[25]: array([ 0.,  0.,  0.,  0.])


In [44]: rand(4)
Out[44]: array([ 0.77109946,  0.08567271,  0.55373
0.51031861])
```

```
In [48]: linspace(1,2,3)
Out[48]: array([ 1. ,  1.5,  2. ])
```

```
In [49]: linspace(1,2,5)
Out[49]: array([ 1.  ,  1.25,  1.5 ,  1.75,  2.  ])
```

```
In [58]: x=[3,4]

In [59]: y=array(x)

In [60]: y
Out[60]: array([3, 4])
```