Data Types and Strings

Manipulating Python data types

D.S. Hwang

Department of Software Science

Dankook University

Outline

Data Types I

Many other kinds of data in the world

- Number
- Text or string

Each of these can be represented as a data type.

- How to manipulate those data types is a big part of being able to program.

Data Types II

Data types in Python

- Built-in type : `int, float, list, str, tuple, dict`
- Data types from the standard library

```
>>> help()

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.6/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".
```

Data Types III

```
16
17 help> keywords
18
19 Here is a list of the Python keywords.  Enter any keyword to get more help.
20
21 False               def                 if                  raise
22 None                del                 import              return
23 True                elif                in                  try
24 and                 else                is                  while
25 as                  except              lambda              with
26 assert              finally             nonlocal            yield
27 break               for                 not
28 class               from                or
29 continue            global              pass
```

`dir()`

- Returns a list of the names in the current scope when there is no argument
- List all the attributes of the argument otherwise

Data Types IV

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__']
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1',
'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan',
'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh
```

`__builtins__` holds all of Python's built-in attributes.

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError'
'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionEr
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellips
'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'Floating
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationEr
...
'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 's
```

Data Types V

The use of underscore (_)

- Names begin and end with two underscores like `__lt__` should not be used.
- Python defines various special methods and variables such as `__init__()`, `__name__`, etc.

```
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
...
'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>> math.__name__
'math'
>>> math.__package__
''
>>> math.__file__
'/Users/dshwang/miniconda3/lib/python3.6/lib-dynload/math.cpython-36m-darwin.so'
```

Data Types VI

The use of a singe underscore

- Used as an identifier
- _ holds the results of the last expression that was evaluated most recently
- Used to ignore the values
- Separate the binary, octal or hex parts of numbers

```
1  >>> 5 + 4
2  9
3  >>> _
4  9
5  >>> _ + 6
6  15
7  >>> a = _
8  >>> a
9  15
10 >>> a, _, b = [1,2,3]
```

Data Types VII

```
11 >>> a
12 1
13 >>> b
14 3
15 >>> _
16 2
17 >>> first, *_, last = range(10)
18 >>> first
19 0
20 >>> last
21 9
22 >>> _
23 [1, 2, 3, 4, 5, 6, 7, 8]
24 >>> million = 1_000_000
25 >>> binary = 0b_0111
26 >>> octa = 0o_64
27 >>> hexa = 0x_23_bc
28 >>> print(million)
29 1000000
30 >>> print(binary)
31 7
32 >>> print(octa)
33 52
```

range(0, 10, 1)

[0, 1, 2, 3,4,5,6,7,8, 9]

0o64

Data Types VIII

```
>>> print(hexa)
9148
```

```
>>> for _ in range(1,10):
...     print("This is an 101 class to learn Python!")
...
This is an 101 class to learn Python!
This is an 101 class to learn Python!
This is an 101 class to learn Python!
This is an 101 class to learn Python!
This is an 101 class to learn Python!
This is an 101 class to learn Python!
This is an 101 class to learn Python!
This is an 101 class to learn Python!
This is an 101 class to learn Python!
```

Integral Data Type

- Two built-in types: `int` vs. `bool`
- Immutable data type
- (Boolean), 0 and `False` are `False`, and any other integer and `True` are `True`
- (Numeric) `True` evaluates to 1 and `False` to 0.

int(True), int(False)
bool(0), bool(-1), bool(1)

Integral Data Type I

Integer

- The size of an integer is limited only by the machine's memory
- Integer literals are written using base 10 (decimal) by default.
- Get used to numeric operators and functions and integer conversion functions

Integral Data Type II

Integer

Figure: Integer conversion functions

Syntax	Description
`x + y`	Adds number `x` and number `y`
`x - y`	Subtracts `y` from `x`
`x * y`	Multiplies `x` by `y`
`x / y`	Divides `x` by `y`; always produces a `float` (or a `complex` if `x` or `y` is `complex`)
`x // y`	Divides `x` by `y`; truncates any fractional part so always produces an `int` result; see also the `round()` function
`x % y`	Produces the modulus (remainder) of dividing `x` by `y`
`x ** y`	Raises `x` to the power of `y`; see also the `pow()` functions
`-x`	Negates `x`; changes `x`'s sign if nonzero, does nothing if zero
`+x`	Does nothing; is sometimes used to clarify code
`abs(x)`	Returns the absolute value of `x`
`divmod(x, y)`	Returns the quotient and remainder of dividing `x` by `y` as a tuple of two `int`s
`pow(x, y)`	Raises `x` to the power of `y`; the same as the `**` operator
`pow(x, y, z)`	A faster alternative to `(x ** y) % z`
`round(x, n)`	Returns `x` rounded to *n* integral digits if *n* is a negative `int` or returns `x` rounded to *n* decimal places if *n* is a positive `int`; the returned value has the same type as `x`; see the text

Integral Data Type III

Integer

Figure: Numeric operators and functions

Syntax	Description
`bin(i)`	Returns the binary representation of `int i` as a string, e.g., `bin(1980) == '0b11110111100'`
`hex(i)`	Returns the hexadecimal representation of `i` as a string, e.g., `hex(1980) == '0x7bc'`
`int(x)`	Converts object `x` to an integer; raises `ValueError` on failure—or `TypeError` if `x`'s data type does not support integer conversion. If `x` is a floating-point number it is truncated.
`int(s, base)`	Converts `str s` to an integer; raises `ValueError` on failure. If the optional *base* argument is given it should be an integer between 2 and 36 inclusive.
`oct(i)`	Returns the octal representation of `i` as a string, e.g., `oct(1980) == '0o3674'`

Integral Data Type I

Boolean

bool()

- Two built-in Boolean objects: `True` and `False`
- The `bool` data type can be called as a function:- with no arguments it returns `False`.
- Programmers using older versions of Python sometimes use 1 and 0 instead of `True` and `False`.
- Python provides three logical operators: `and`, `or`, and `not`

Integral Data Type II

Boolean

```
1  >>> check = bool()
2  >>> check
3  False
4  >>> t = True
5  >>> f = False
6  >>> t and f
7  False
8  >>> t and check
9  False
10 >>> not f
11 True
```

Floating-Point Data Type I

- Three kinds of floating-point values
  1. `float`
  2. `complex`
  3. `decimal.Decimal` from the standard library.
- These are immutable!
- Computers natively represent floating-point numbers using base 2.
- So, some decimals can be represented exactly (such as 0.5), but others only approximately (such as 0.1 and 0.2).

```
1 >>> -0.003333, 4.5, -2.5, 8.9e-4 # Python 3.1 later
2 (-0.003333, 4.5, -2.5, 0.00089)
3 >>> 0.0, 5.4, -2.5, 8.9e-4                # Python 2.7
4 (0.0, 5.400000000000004, -2.5, 0.0008899999999999995)
```

Floating-Point Number I

float(), float(str)

- The float data type can be called as a function—with no arguments it returns 0.0.
- When used for conversions a string argument can be given, either using simple decimal notation or using exponential notation.
- `float` returns `NaN` (”not a number”) or ”infinity”.

Floating-Point Number II

Figure: Numeric operators and functions

Syntax	Description
`math.acos(x)`	Returns the arc cosine of `x` in radians
`math.acosh(x)`	Returns the arc hyperbolic cosine of `x` in radians
`math.asin(x)`	Returns the arc sine of `x` in radians
`math.asinh(x)`	Returns the arc hyperbolic sine of `x` in radians
`math.atan(x)`	Returns the arc tangent of `x` in radians
`math.atan2(y, x)`	Returns the arc tangent of `y / x` in radians
`math.atanh(x)`	Returns the arc hyperbolic tangent of `x` in radians
`math.ceil(x)`	Returns $\lceil x \rceil$, i.e., the smallest integer greater than or equal to `x` as an `int`; e.g., `math.ceil(5.4) == 6`
`math.copysign(x,y)`	Returns `x` with `y`'s sign
`math.cos(x)`	Returns the cosine of `x` in radians
`math.cosh(x)`	Returns the hyperbolic cosine of `x` in radians
`math.degrees(r)`	Converts `float r` from radians to degrees
`math.e`	The constant e; approximately 2.718 281 828 459 045 1
`math.exp(x)`	Returns e^x, i.e., `math.e ** x`
`math.fabs(x)`	Returns $\lvert x \rvert$, i.e., the absolute value of `x` as a `float`
`math.factorial(x)`	Returns $x!$
`math.floor(x)`	Returns $\lfloor x \rfloor$, i.e., the largest integer less than or equal to `x` as an `int`; e.g., `math.floor(5.4) == 5`
`math.fmod(x, y)`	Produces the modulus (remainder) of dividing `x` by `y`; this produces better results than `%` for `floats`
`math.frexp(x)`	Returns a 2-tuple with the mantissa (as a `float`) and the exponent (as an `int`) so, $x = m \times 2^e$; see `math.ldexp()`

Syntax	Description
`math.frexp(x)`	Returns a 2-tuple with the mantissa (as a `float`) and the exponent (as an `int`) so, $x = m \times 2^e$; see `math.ldexp()`
`math.fsum(i)`	Returns the sum of the values in iterable `i` as a `float`
`math.hypot(x, y)`	Returns $\sqrt{x^2 + y^2}$
`math.isinf(x)`	Returns `True` if `float x` is $\pm$ `inf` ($\pm\infty$)
`math.isnan(x)`	Returns `True` if `float x` is `nan` ("not a number")
`math.ldexp(m, e)`	Returns $m \times 2^e$; effectively the inverse of `math.frexp()`
`math.log(x, b)`	Returns $\log_b x$; b is optional and defaults to `math.e`
`math.log10(x)`	Returns $\log_{10} x$
`math.log1p(x)`	Returns $\log_e(1+x)$; accurate even when x is close to 0
`math.modf(x)`	Returns `x`'s fractional and whole parts as two `floats`
`math.pi`	The constant π; approximately 3.141 592 653 589 793 1
`math.pow(x, y)`	Returns x^y as a `float`
`math.radians(d)`	Converts `float d` from degrees to radians
`math.sin(x)`	Returns the sine of `x` in radians
`math.sinh(x)`	Returns the hyperbolic sine of `x` in radians
`math.sqrt(x)`	Returns $\sqrt{x}$
`math.tan(x)`	Returns the tangent of `x` in radians
`math.tanh(x)`	Returns the hyperbolic tangent of `x` in radians
`math.trunc(x)`	Returns the whole part of `x` as an `int`; same as `int(x)`

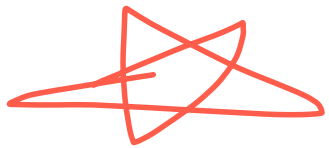
Floating-Point Number III

```
1  >>> s = 14.25.hex() # string in hexadecimal
2  >>> s
3  '0x1.c800000000000p+3'
4  >>> f = float.fromhex(s)
5  >>> f
6  14.25
7  >>> t = f.hex()
8  >>> t
9  '0x1.c800000000000p+3'
10 >>> import math
11 >>> math.pi*  5 ** 2
12 78.53981633974483
13 >>> math.hypot(5,12)
14 13.0
15 >>> math.modf(13.732)
16 (0.7319999999999993, 13.0)
```

class = data + method

object.data
object.f()

Decimal Number I



- In many applications the *numerical inaccuracies* that can occur when using `float`s don't matter.
- Sometimes numerical inaccuracies are important.

```
1 >>> import decimal
2 >>> v = decimal.Decimal(12345678999999999999)
3 >>> v
4 Decimal('123456789999999999')
5 >>> w = decimal.Decimal(99999999999999999999)
6 >>> v + w
7 Decimal('1012345678999999999998')
```

Mutable vs. Immutable I

i = 1000

- Everything in Python is an object.
- Every variable holds an object instance since everything in Python is an Object.
- When an object is initiated, it is assigned a unique object id and Its type is defined at runtime and once set can never change.
- The content of a data type can be changed if it is mutable.
- A mutable object can be changed after it is created, and an immutable object can't.
- Objects of built-in types like (`int, float, bool, str, tuple, unicode`) are immutable.

Mutable vs. Immutable II

- Objects of built-in types like (`list, set, dict`) are mutable.
- Custom classes are generally mutable.

Figure: Immutable data types

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Mutable vs. Immutable III

`id` and `type`

- The built-in function `id` returns the identity of an object as an integer.
- This integer usually corresponds to the object's location in memory.
- The built-in function `type()` returns the type of an object.
- The `is` operator compares the identity of two objects.

Mutable vs. Immutable IV

```
>>> msg1 = 'You can do better!'
>>> msg2 = 'You can do better!'
>>> id(msg1)
4367858832
>>> id(msg2)
4367858688
>>> print(msg1 is msg2)
False
>>> print(msg1 == msg2)
True
>>> msg1[3]='x'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> type(msg1)
<class 'str'>
>>> msg2 = 'New'
>>> id(msg2)
4367853584
>>> print(msg1 == msg2)
False
```

char* str = "Dankook"
str[3]=?

Mutable vs. Immutable V

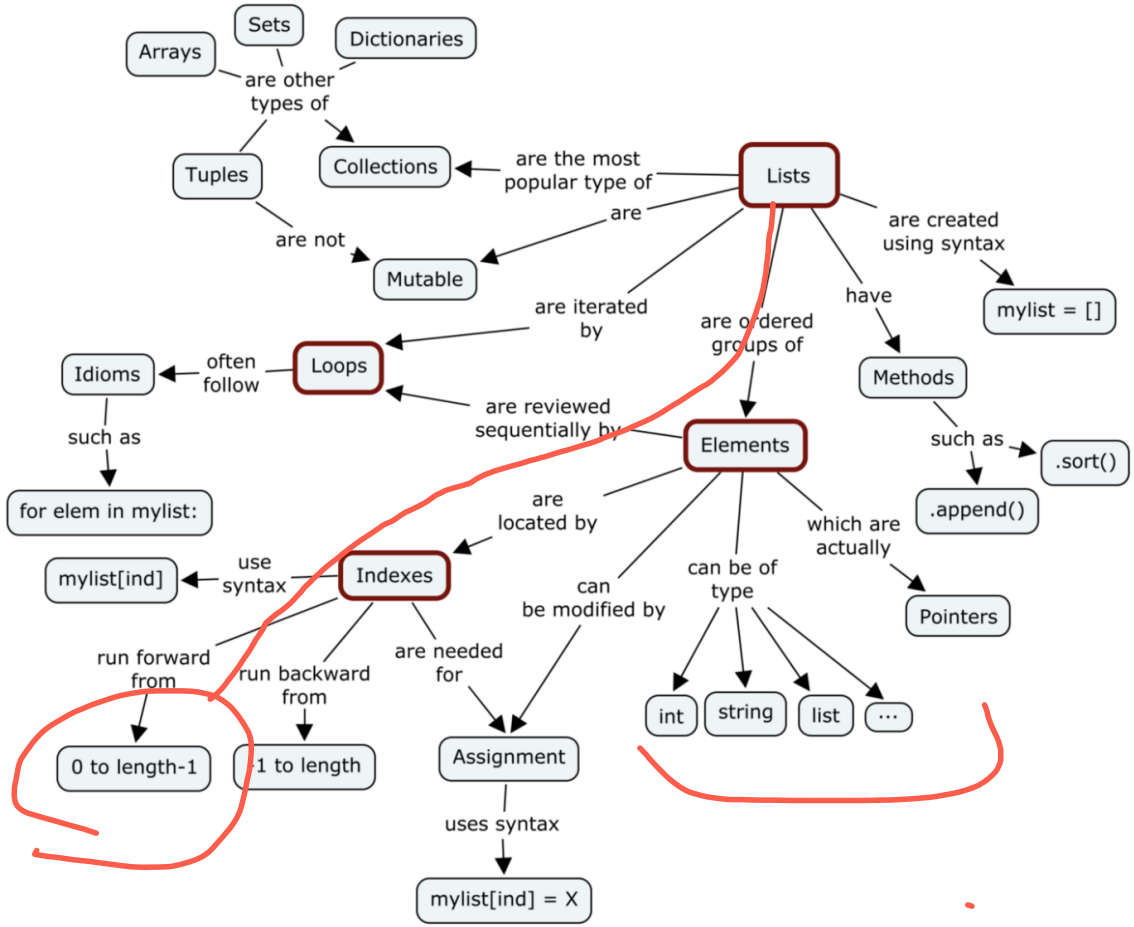
How to check out if a variable is immutable:

- Check the identity of variable or value

```
>>> y = 10
>>> x = y
>>> id(y)
4363327088
>>> id(x)
4363327088
>>> id(10)    # object 10 was never modified.
4363327088
>>> x = x + 1
>>> x
11
>>> id(x)        # different id
4363327120
>>> id(x) != id(y)
True
>>> id(x) != id(10)
True
```

Mutable vs. Immutable VI

Figure: Python data types



String Data Type I

Computers may have been invented to do arithmetic, but these days, most of them spend a lot of their time processing text.

- desktop chat programs
- Web surfing
- E-mail
- Word processing
- ...

Text

- A piece of text is represented as a string.
- A string is a sequence of characters (letters, numbers, and symbols).

String Data Type II

Computer don't understand characters:

- Each character is encoded with one bye (0 to 255).
- Refer to "8 bits of memory as a byte of memory"
- The `ord()` function tells us the number value of a simple ASCII character.
- ASCII, American Standard Code for Information Interchange
- By the 80s, we assumed one byte was one character due to ASCII.

String Data Type III

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

ASCII 65

A

alt + 65
(Capital letter A)

most consulted

ñ	**énye, n with tilde** (alt + 164)
■	**black square** (alt + 254)
²	**superscript two, square** (alt + 253)
º	**degree symbol** (alt + 248)
'	**apostrophe, single quote** (alt + 39)
µ	**letter Mu, micro, micron** (alt + 230)
©	**copyright symbol** (alt + 184)
®	**registered trademark** (alt + 169)
³	**superscript three, cube** (alt + 252)
á	**a with acute accent** (alt + 160)

frequently-used (spanish language)

ñ	alt + 164
Ñ	alt + 165
@	alt + 64
¿	alt + 168
?	alt + 63
¡	alt + 173
!	alt + 33
:	alt + 58
/	alt + 47
\	alt + 92

vowels acute accent (spanish language)

á	alt + 160
é	alt + 130
í	alt + 161
ó	alt + 162
ú	alt + 163
Á	alt + 181
É	alt + 144
Í	alt + 214
Ó	alt + 224
Ú	alt + 233

vowels with diaresis

ä	alt + 132
ë	alt + 137
ï	alt + 139
ö	alt + 148
ü	alt + 129
Ä	alt + 142
Ë	alt + 211
Ï	alt + 216
Ö	alt + 153
Ü	alt + 154

mathematical symbols

½	alt + 171
¼	alt + 172
¾	alt + 243
¹	alt + 251
³	alt + 252
²	alt + 253
ƒ	alt + 159
±	alt + 241
×	alt + 158
÷	alt + 246

commercial / trade symbols

$	alt + 36
£	alt + 156
¥	alt + 190
¢	alt + 189
¤	alt + 207
®	alt + 169
©	alt + 184
ª	alt + 166
º	alt + 167
°	alt + 248

quotes and parenthesis

"	alt + 34
'	alt + 39
(	alt + 40
)	alt + 41
[	alt + 91
]	alt + 93
{	alt + 123
}	alt + 125
«	alt + 174
»	alt + 175

Multi-Byte Representation I

- Computers must represent the wide range of characters we are using.
  - UTF-16 : Fixed length (2 bytes)
  - UTF-32 : Fixed length (4 bytes)
  - UTF-8 : 1 to 4 bytes
    - Compatible with ASCII
    - Automatic detection between ASCII and UTF-8
    - UTF-8 is recommended for encoding data to be exchanged between systems

Strings I

```
>>> s = 'Dankook' # ver. 2.7.15
>>> type(s)
<type 'str'>
>>> s = u'Dankook'
>>> type(s)
<type 'unicode'>
```

```
>>> s = 'Dankook' # ver. 3.6.7
>>> type(s)
<class 'str'>
>>> s = u'Dankook'
>>> type(s)
<class 'str'>
```

Strings II

Define a string :

- Python 2 has ASCII `str()` types, separate `unicode()`, but no byte type.
- Python 3 supports Unicode (utf-8) strings, and 2 byte classes: `byte` and `bytearrays`.
- Use single or double quotation in representing strings
- In Python 3.xx, all strings are based on UTF-8.

Strings

Join two strings

```
1 >>> 'Albert' 'Einstein'
2 'AlbertEinstein'
3 >>> 'Albert_' 'Einstein'
4 'Albert Einstein'
5 >>> 'Albert' + ' Einstein'
6 'Albert Einstein'
7 >>> "" # empty string
8 ''
```

Overloaded operator

- The $+$ operator is used for both numeric addition and for string concatenation

Strings

Repeat two strings

```
1 >>> 'AT' * 5
2 'ATATATATAT'
3 >>> 4 * '-'
4 '----'
5 >>> 'GC' * 0
6 ''
7 >>> 'TATATATA' * -3
8 ''
```

- The * operator concatenates a string by the number
- If the integer is less than or equals to zero, then this operator yields the empty string

Escape Characters

```
1 >>> "that's better"
2 "that's better"
3 >>> 'She said, "That' + "'" + 's hard to read."'
4 'She said, "That\'s hard to read."'
```

- The combination of the backslash and the single quote
- escape from Python's usual syntax rules for a moment

Escape Characters

Figure: String escape

Escape	Meaning
newline	Escape (i.e., ignore) the newline
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII bell (BEL)
\b	ASCII backspace (BS)
\f	ASCII formfeed (FF)
\n	ASCII linefeed (LF)
\N{*name*}	Unicode character with the given name
ooo	Character with the given octal value
\r	ASCII carriage return (CR)
\t	ASCII tab (TAB)
\u*hhhh*	Unicode character with the given 16-bit hexadecimal value
\U*hhhhhhhh*	Unicode character with the given 32-bit hexadecimal value
\v	ASCII vertical tab (VT)
\x*hh*	Character with the given 8-bit hexadecimal value

Multiline Strings

```
1 >>> 'I do my best
2 SyntaxError: EOL while scanning string literal
3 >>> '''I do
4 my
5 best'''
6 'I do\nmy\nbest'
```

- If you create a string using single or double quotes, the whole string must fit onto a single line.
- To span multiple lines, put three single quotes or three double quotes around the string instead of one of each.
- The major operating systems use a different set of characters to indicate the end of a line.
  - MS Windows: \r\n
  - Linux, Mac OS Leopard or Unix: \n
  - Mac OS X: \r

Print

```
1 >>> print('one\ttwo\nthree\tfour')
2 one     two
3 three   four
4 >>> area = 3.14159 * 5 * 5
5 >>> print("The area of the circle is" , area, "sq cm.")
6 The area of the circle is 78.53975 sq cm.
7 >>> print("The area of the circle is %f sq cm." % area)
8 The area of the circle is 78.539750 sq cm.
```

Conversion specifier

- %f formates the float-point number
- %d formates the integer value
- %s formates the string

Print

```
1  >>> line = input()
2  Galapagos Islands
3  >>> print(line)
4  Galapagos Islands
5  >>> line = input()
6  123
7  >>> value = input()
8  123
9  >>> value = int(value)
10 >>> print(value * 2)
11 246
```

- `input` reads a single line of text from the keyboard.
- If you are expecting the user to enter a number, you must use int or float to convert the string to the required type.

Print

float()

```
1 >>> value = float(input())
2 abc
3 Traceback (most recent call last):
4   File "<stdin>", line 1, in <module>
5 ValueError: could not convert string to float: 'abc'
6 >>> name = input("Please enter a name: " )
7 Please enter a name: Darwin
8 >>> print(name)
9 Darwin
```

- `input` can be given a string argument, which is used to prompt the user for input.

Summary

- String and various generating methods
- ASCII vs. unicode
- Escape sequence
- Formatted printing
- How to input data from a user

Programming challenging!

Alan Perlis, computer scientist, 1922-1990 (First recipient of the Turing Award (1966))

- ” You think you know when you can learn, are more sure when you can write, even more when you can teah, but certain when you can program.”
- ”Within a computer, natural language is unnatural.”
- ”To understand a program, you must become both the machine and the program.”

Problem 1

Practice how variables refer to Python objects. Test each code line to understand the results of Python objects.

1. `a = -11`
2. `b = 11`
3. `c = 9.0`
4. `d = b/ a`
5. `e = c/a`
6. `s = 'b / a = %g' % (b / a)`

Problem 2

Convert between object types

1. `a = 3`
2. `b = float(a)`
3. `c = 3.9`
4. `d = int( c )`
5. `e = round( c )`
6. `f = int(round(c))`
7. `d = str(c)`
8. `e = '-4.2'`
9. `f = float(e)`

Problem 3

Convert between object types

1. `a = 3`
2. `b = float(a)`
3. `c = 3.9`
4. `d = int( c )`
5. `e = round( c )`
6. `f = int(round(c))`
7. `d = str(c)`
8. `e = '-4.2'`
9. `f = float(e)`

Problem 4

Compute the equation.

$$\sinh(x) = \frac{1}{2}(e^x - e^{-x})$$

- import `math` and use `sinh, pi, e, exp`

Problem 5

Throw a ball with velocity v_0, at an angle θ with the horizontal, from the point($x = 0, y = y_0$). The trajectory of the all is a parabola. We neglect air resistance.

$$y = x \tan\theta - \frac{1}{2v_0} \frac{gx^2}{\cos^2\theta} + y_0$$

- import `math` and use `cos`
- Initialize input data (v_0, θ, y_0), where v_0's unit is km/h and θ is degree
- g takes $9.81 m/s^2$.
- Compute and print y

Problem 6

Let p be a bank's interest rate in percent per year. After n years, an initial amount A has taken grown to

$$A(1 + \frac{p}{100})^n.$$

- Make a program for computing how much money $10,000,000$ have grown to after three years with 5% interest rate.