Introduction to Computer Science

Basic Types

鄒年棣 (Nien-Ti Tsou)

Numeric types

- The numeric type `int`, `float`, and `complex`.
- The statement `k = 3` assigns the variable `k` to an integer
- Applying an operation of the type `+`, `-`, or `*` to integers returns an integer.
- The division operator, `//`, returns an integer, while `/` may return a `float`
- The set of integers in Python is unbounded; there is no largest integer. The limitation here is the computer's memory rather than any fixed value given by the language.

Floating point numbers

- If you execute the statement `a = 3.0` or `a = 30.0e-1`, you create a floating-point number (Python type: `float`). The expression reads in mathematical notation $a = 30.0 \times 10^{-1}$.

- Applying the elementary mathematical operations `+`, `-`, `*`, and `/` to two floating-point numbers or to an integer and a floating-point number returns a floating-point number.

- Operations between floating-point numbers rarely return the exact result expected from rational number operations:

```
0.4 - 0.3
```

- This facts matters, when comparing floating point numbers:

```
Difference = 0.4 - 0.3
print(Difference == 0.1)
```

```
True
```

Floating point numbers

- If you execute the statement `a = 3.0` or `a = 30.0e-1`, you create a floating-point number (Python type: `float`). The expression reads in mathematical notation $a = 30.0 \times 10^{-1}$.

- Applying the elementary mathematical operations `+`, `-`, `*`, and `/` to two floating-point numbers or to an integer and a floating-point number returns a floating-point number.

- Operations between floating-point numbers rarely return the exact result expected from rational number operations:

```
0.4 - 0.3
```

```
0.10000000000000003
```

- This facts matters, when comparing floating point numbers:

```
Difference = 0.4 - 0.3
print(Difference == 0.1)
```

```
True
```

Floating point numbers

- If you execute the statement `a = 3.0` or `a = 30.0e-1`, you create a floating-point number (Python type: `float`). The expression reads in mathematical notation $a = 30.0 \times 10^{-1}$.

- Applying the elementary mathematical operations `+`, `-`, `*`, and `/` to two floating-point numbers or to an integer and a floating-point number returns a floating-point number.

- Operations between floating-point numbers rarely return the exact result expected from rational number operations:

```
0.4 - 0.3
```

```
0.10000000000000003
```

- This facts matters, when comparing floating point numbers:

```
Difference = 0.4 - 0.3
print(Difference == 0.1)
```

```
False
True
```

Floating point numbers

- If you execute the statement `a = 3.0` or `a = 30.0e-1`, you create a floating-point number (Python type: `float`). The expression reads in mathematical notation $a = 30.0 \times 10^{-1}$.

- Applying the elementary mathematical operations `+`, `-`, `*`, and `/` to two floating-point numbers or to an integer and a floating-point number returns a floating-point number.

- Operations between floating-point numbers rarely return the exact result expected from rational number operations:

```
0.4 - 0.3
```

```
0.10000000000000003
```

- This facts matters, when comparing floating point numbers:

```
Difference = 0.4 - 0.3
print(Difference == 0.1)
print(abs(Difference - 0.1) < 1e-6)
```

```
False
True
```

Floating point representation

- Internally, floating-point numbers are represented by four quantities: the sign, the mantissa, the exponent sign, and the exponent:

$$\mathrm{sign}(x)\left(x_0 + x_1\beta^{-1} + \ldots + x_{t-1}\beta^{-(t-1)}\right)\beta^{(e)|e|}$$

- $x_0 \ldots x_{t-1}$ is called the mantissa with basis $\beta = 2$ on a typical Intel processor, e the exponent. t is called the mantissa length.

- To represent a number in the `float` type 64 bits are used: 2 bits for the signs, $t = 52$ bits for the mantissa and 10 bits for the exponent `|e|`. The upper bound for the exponent is $2^{10}-1 = 1023$.

$$\mathrm{fl}_{\min} = 1.0 \times 2^{-1023} \approx 10^{-308}$$

Binary

$$\mathrm{fl}_{\max} = 1.111\ldots1 \times 2^{1023} \approx 10^{308}$$

Binary

Infinite and not a number

- Sometimes floating-point numbers are outside their range, giving the special floating-point number `inf`.
- Working with `inf` may lead to mathematically undefined results, giving not-a-number `nan`.
- They can also be created by `float('inf')` & `float('nan')`

```
a = 1.1111 * 2**1023
a = a * 2
b = float('inf')

print(a)
print(a+b)
print(a-b)
print(a/b)


c = float('nan')
print(c)
```

Infinite and not a number

- Sometimes floating-point numbers are outside their range, giving the special floating-point number `inf`.
- Working with `inf` may lead to mathematically undefined results, giving not-a-number `nan`.
- They can also be created by `float('inf')` & `float('nan')`

```
a = 1.1111 * 2**1023
a = a * 2
b = float('inf')

print(a)
print(a+b)
print(a-b)
print(a/b)


c = float('nan')
print(c)
```

```
inf
inf
nan
nan
```

Infinite and not a number

- Sometimes floating-point numbers are outside their range, giving the special floating-point number `inf`.
- Working with `inf` may lead to mathematically undefined results, giving not-a-number `nan`.
- They can also be created by `float('inf')` & `float('nan')`

```
a = 1.1111 * 2**1023
a = a * 2
b = float('inf')

print(a)
print(a+b)
print(a-b)
print(a/b)


c = float('nan')
print(c)
```

```
inf
inf
nan
nan
```

```
nan
```

Complex numbers

- Complex numbers consist of two floating-point numbers, the real part *a* of the number and its imaginary part *b*. It is written as $z=a+bi$, where i defined by $i^2 = -1$ is the imaginary unit.
- A complex number is formed by the sum of a floating- point number and an imaginary number, for example, `z = 3.5 + 5.2j`.
- The Python way of expressing an imaginary number is not a product: `j` is just a suffix to indicate that the number is imaginary.

```
b = 5.2
z = b*1j
z = bj
z = b*j
```

```
5.2j
```

```
NameError: name        is not defined
```

```
NameError: name        is not defined
```

Complex numbers

- Complex numbers consist of two floating-point numbers, the real part *a* of the number and its imaginary part *b*. It is written as $z=a+bi$, where i defined by $i^2 = -1$ is the imaginary unit.
- A complex number is formed by the sum of a floating- point number and an imaginary number, for example, `z = 3.5 + 5.2j`.
- The Python way of expressing an imaginary number is not a product: `j` is just a suffix to indicate that the number is imaginary.

```
b = 5.2
z = b*1j
z = bj
z = b*j
```

```
5.2j
```

```
NameError: name 'bj' is not defined
```

```
NameError: name      is not defined
```

Complex numbers

- Complex numbers consist of two floating-point numbers, the real part *a* of the number and its imaginary part *b*. It is written as $z=a+bi$, where i defined by $i^2 = -1$ is the imaginary unit.
- A complex number is formed by the sum of a floating- point number and an imaginary number, for example, `z = 3.5 + 5.2j`.
- The Python way of expressing an imaginary number is not a product: `j` is just a suffix to indicate that the number is imaginary.

```
b = 5.2
z = b*1j
z = bj
z = b*j
```

```
5.2j
```

```
NameError: name 'bj' is not defined
```

```
NameError: name 'j' is not defined
```

Complex numbers

- The method `conjugate` returns the conjugate of `z`:

```
z = 3.2 + 5.2j
print(z.conjugate())
```

```
(3.2-5.2j)
```

- One may access the real and imaginary parts of a complex number *z* using the `real` and `imag` attributes. Those attributes are read-only.

```
z = 1j
print(z.real)
print(z.imag)
z.imag = 2
```

```
0.0
1.0
```

```
AttributeError: readonly attribute
```

- It cannot convert a complex number to a real number:

```
z = 1 + 0j
print(z == 1)
a = float(z)
```

```

```

```
TypeError: can't convert complex to float
```

Complex numbers

- The method `conjugate` returns the conjugate of `z`:

```
z = 3.2 + 5.2j
print(z.conjugate())
```

```
(3.2-5.2j)
```

- One may access the real and imaginary parts of a complex number *z* using the `real` and `imag` attributes. Those attributes are read-only.

```
z = 1j
print(z.real)
print(z.imag)
z.imag = 2
```

```
0.0
1.0
```

```
AttributeError: readonly attribute
```

- It cannot convert a complex number to a real number:

```
z = 1 + 0j
print(z == 1)
a = float(z)
```

```
True
```

```
TypeError: can't convert complex to float
```

Boolean

- We introduced before.
- In fact, Booleans and integers are the same.
- The only difference is in the representation of numbers 0 and 1 which is in the case of Booleans `False` and `True` respectively.

Strings

- The type `string` is a type used for text.
- A string is enclosed either by single or double quotes. If a string contains several lines, it has to be enclosed by three double quotes `"""` or three single quotes `'''`
- A multi line string automatically includes `'\n'`.

```
name = 'Johan Carlsson'
child = "Åsa is Johan Carlsson's daughter"
book = """Aunt Julia
and the Scriptwriter"""
print(name)
print(child)
print(book)
```

Print and the content are slightly different

```
Johan Carlsson
Åsa is Johan Carlsson's daughter
Aunt Julia
and the Scriptwriter
```

```
In [2]: book
```

Strings

- The type `string` is a type used for text.
- A string is enclosed either by single or double quotes. If a string contains several lines, it has to be enclosed by three double quotes `"""` or three single quotes `'''`
- A multi line string automatically includes `'\n'`.

```
name = 'Johan Carlsson'
child = "Åsa is Johan Carlsson's daughter"
book = """Aunt Julia
and the Scriptwriter"""
print(name)
print(child)
print(book)
```

```
Johan Carlsson
Åsa is Johan Carlsson's daughter
Aunt Julia
and the Scriptwriter
```

Print and the content are slightly different

```
In [2]: book
Out[2]: 'Aunt Julia\nand the
Scriptwriter'
```

More about string

- Strings can be indexed with simple indexes or slices

```
print(book[-1])
print(book[-12:])
print(book[-12:-1])
```

- Strings are immutable; that is, items cannot be altered.

```
book[1] = 'a'
```

```
TypeError: 'str' object does not support
item assignment
```

- The string `'\n'` is used to insert a line break and `'\t'` inserts a horizontal tabulator (TAB) into the string to align several lines

```
print('Temperature:\t20\tC\nPressure:\t5\tPa')
```

```
Temperature:    20      C
Pressure:       5       Pa
```

More about string

- Strings can be indexed with simple indexes or slices

```
print(book[-1])
print(book[-12:])
print(book[-12:-1])
```

```
r
```

- Strings are immutable; that is, items cannot be altered.

```
book[1] = 'a'
```

```
TypeError: 'str' object does not support
item assignment
```

- The string `'\n'` is used to insert a line break and `'\t'` inserts a horizontal tabulator (TAB) into the string to align several lines

```
print('Temperature:\t20\tC\nPressure:\t5\tPa')
```

```
Temperature:    20      C
Pressure:       5       Pa
```

More about string

- Strings can be indexed with simple indexes or slices

```
print(book[-1])
print(book[-12:])
print(book[-12:-1])
```

```
r
Scriptwriter
```

- Strings are immutable; that is, items cannot be altered.

```
book[1] = 'a'
```

```
TypeError: 'str' object does not support
item assignment
```

- The string `'\n'` is used to insert a line break and `'\t'` inserts a horizontal tabulator (TAB) into the string to align several lines

```
print('Temperature:\t20\tC\nPressure:\t5\tPa')
```

```
Temperature:    20      C
Pressure:       5       Pa
```

More about string

- Strings can be indexed with simple indexes or slices

```
print(book[-1])
print(book[-12:])
print(book[-12:-1])
```

```
r
Scriptwriter
Scriptwrite
```

- Strings are immutable; that is, items cannot be altered.

```
book[1] = 'a'
```

```
TypeError: 'str' object does not support
item assignment
```

- The string `'\n'` is used to insert a line break and `'\t'` inserts a horizontal tabulator (TAB) into the string to align several lines

```
print('Temperature:\t20\tC\nPressure:\t5\tPa')
```

```
Temperature:    20      C
Pressure:       5       Pa
```

Escape sequence

- The backslash (`\`) character is used to escape characters that otherwise have a special meaning, such as newline (`\n`), backslash itself (`\\`), or the quote character(`\"`).

```
print('\n')
print('\\')
print('\"')
print('\'')
```

```
\
"
'
```

- String literals may optionally be prefixed with a letter `'r'` or `'R'`; such strings are called raw strings and use different rules for backslash escape sequences.

- `r"\"` is not a valid string literal, since the backslash would escape the following quote character

```
print(r"\n")
print(r'\n')
print(r"\"")
print('\\')
print(r"\")
```

```
\n
\n
\"
\
```

```
SyntaxError: EOL while scanning string literal
```

Operations on strings and string method

- Addition of strings:

```
last_name = 'Carlsson'
first_name = 'Johanna'
full_name = first_name + ' ' + last_name

print(full_name)
print(first_name,last_name)
print(first_name,' ',last_name)
print(first_name,'\t',last_name)
```

```
Johanna Carlsson
```

- Multiplication is just repeated addition:

```
game = 2 * 'Yo'
print(game)
```

```
YoYo
```

Operations on strings and string method

- Addition of strings:

```
last_name = 'Carlsson'
first_name = 'Johanna'
full_name = first_name + ' ' + last_name

print(full_name)
print(first_name,last_name)
print(first_name,' ',last_name)
print(first_name,'\t',last_name)
```

```
Johanna Carlsson
Johanna Carlsson
```

- Multiplication is just repeated addition:

```
game = 2 * 'Yo'
print(game)
```

```
YoYo
```

Operations on strings and string method

- Addition of strings:

```
last_name = 'Carlsson'
first_name = 'Johanna'
full_name = first_name + ' ' + last_name

print(full_name)
print(first_name,last_name)
print(first_name,' ',last_name)
print(first_name,'\t',last_name)
```

```
Johanna Carlsson
Johanna Carlsson
Johanna   Carlsson
```

- Multiplication is just repeated addition:

```
game = 2 * 'Yo'
print(game)
```

```
YoYo
```

Operations on strings and string method

- Addition of strings:

```
last_name = 'Carlsson'
first_name = 'Johanna'
full_name = first_name + ' ' + last_name

print(full_name)
print(first_name,last_name)
print(first_name,' ',last_name)
print(first_name,'\t',last_name)
```

```
Johanna Carlsson
Johanna Carlsson
Johanna 	Carlsson
```

- Multiplication is just repeated addition:

```
game = 2 * 'Yo'
print(game)
```

```
YoYo
```

ASCII code

- ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard. ASCII codes represent text in computers.

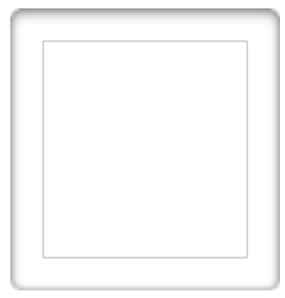
```
c = 'a'
print(ord(c))

c = 97
print(chr(c))
```

```
97
a
```

- When strings are compared, lexicographical order applies and the uppercase form less than the lowercase form of the same letter

```
print('Anna' > 'Arvi')
print('ANNA' < 'anna')
print('10B' < '11A')
```



ASCII code

- ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard. ASCII codes represent text in computers.

```
c = 'a'
print(ord(c))

c = 97
print(chr(c))
```

```
97
a
```

- When strings are compared, lexicographical order applies and the uppercase form less than the lowercase form of the same letter

```
print('Anna' > 'Arvi')
print('ANNA' < 'anna')
print('10B' < '11A')
```

```
False
```

ASCII code

- ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard. ASCII codes represent text in computers.

```
c = 'a'
print(ord(c))

c = 97
print(chr(c))
```

```
97
a
```

- When strings are compared, lexicographical order applies and the uppercase form less than the lowercase form of the same letter

```
print('Anna' > 'Arvi')
print('ANNA' < 'anna')
print('10B' < '11A')
```

```
False
True
```

ASCII code

- ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard. ASCII codes represent text in computers.

```
c = 'a'
print(ord(c))

c = 97
print(chr(c))
```

```
97
a
```

- When strings are compared, lexicographical order applies and the uppercase form less than the lowercase form of the same letter

```
print('Anna' > 'Arvi')
print('ANNA' < 'anna')
print('10B' < '11A')
```

```
False
True
True
```

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Splitting and join strings

- This method generates a list from a string by using a single or multiple blanks as separators. Alternatively, an argument can be given by specifying a particular string as a separator

```
text = 'quod erat demonstrandum'
print(text.split())
table = 'Johan;Carlsson;19890327'
print(table.split(';'))
king = 'CarlXVIGustaf'
print(king.split('XVI'))
```

```
['quod', 'erat', 'demonstrandum']
['Johan', 'Carlsson', '19890327']
['Carl', 'Gustaf']
```

- Joining a list to a string is the reverse operation of splitting:

```
sep = '; '
print(sep.join(['Johan','Carlsson','19890327']))
```

```
Johan; Carlsson; 19890327
```

Searching and replacing in a string

- This method returns the first index in the string, where a given search substring starts. If the search string is not found, the return value of the method is -1.

```
birthday = '20101210'
print(birthday.find('10'))
print(birthday.find('@@'))
```

```
2
-1
```

- The `replace()` function returns a new string, and will replace all occurrences of the input.

```
line = 'the quick brown fox jumped over a lazy dog'
line2 = line.replace('o', '--')
print(line2)
```

```
'the quick br--wn f--x jumped --ver a lazy d--g'
```

String formatting

- String formatting is done using the `format` method:

```
course_code = 'Intro to CS'
print("This course's name is {}".format(course_code))
```

```
This course's name is Intro to CS
```

- The function `format` is a string method; it scans the string for the occurrence of placeholders, which are enclosed by curly brackets. These placeholders are replaced by the variables you specified. Format specifications are indicated by a colon, `":"`, as their prefix. For the `float` type, we can use `{:f}` or `{:e}`:

```
quantity = 33.45
print("{:f}".format(quantity))
print("{:1.1f}".format(quantity))
print("{:.2e}".format(quantity))

print("{:5.1f}".format(quantity))
```

```
33.450000
33.5
3.35e+01

```

String formatting

- String formatting is done using the `format` method:

```
course_code = 'Intro to CS'
print("This course's name is {}".format(course_code))
```

```
This course's name is Intro to CS
```

- The function `format` is a string method; it scans the string for the occurrence of placeholders, which are enclosed by curly brackets. These placeholders are replaced by the variables you specified. Format specifications are indicated by a colon, `":"`, as their prefix. For the `float` type, we can use `{:f}` or `{:e}`:

```
quantity = 33.45
print("{:f}".format(quantity))
print("{:1.1f}".format(quantity))
print("{:.2e}".format(quantity))

print("{:5.1f}".format(quantity))
```

```
33.450000
33.5
3.35e+01
```

```
 33.5
```

String formatting

- The format specifiers allow to specify digits following the decimal point in the representation. The total number of symbols including leading blanks.
- The first {} pair is replaced by the first argument and the following pairs by the subsequent arguments. Alternatively, it may also be convenient to use the key-value syntax:

```
print("{name} {value:.1f}".format(name="quantity",value=quantity))
```

```
quantity 33.5
```

```
print('First: {first}. Last: {last}.'.format(last='Z', first='A'))
```

```
First: A. Last: Z.
```

```
print('First: {1}. Last: {2}.'.format('B','A','Z'))
```

```

```

String formatting

- The format specifiers allow to specify digits following the decimal point in the representation. The total number of symbols including leading blanks.
- The first {} pair is replaced by the first argument and the following pairs by the subsequent arguments. Alternatively, it may also be convenient to use the key-value syntax:

```
print("{name} {value:.1f}".format(name="quantity",value=quantity))
```

```
quantity 33.5
```

```
print('First: {first}. Last: {last}.'.format(last='Z', first='A'))
```

```
First: A. Last: Z.
```

```
print('First: {1}. Last: {2}.'.format('B','A','Z'))
```

```
First: A. Last: Z.
```