Python 101

CS*101* `lec02`

Data types and Boolean logic

2019-09-16

Announcements

quiz: `quiz02` due on Tues 9/17

lab: `lab01` on Fri 9/20

hw: `hw01` due TODAY (Mon 9/16)

`Office Hours : Every Monday and Wednesday`
6.30pm - 7.30pm at the cafe beside the library

Recap

Variable Question

```
x = 10 + 3
y = x
x = 5
```

What is the value of `y`?

A 13

B 10

C 5

Question

```
x = 10 + 3
y = x
x = 5
```

What is the value of `y`?

A 13 ⋆⋆

B 10

C 5

Components

Literals — `4, 11.8`

Components

Literals — `4, 11.8`

Operators —

Components

Literals — `4, 11.8`

Operators — `+`, `*`

Components

Literals — `4, 11.8`

Operators — `+`, `*`

Variables — `x`, `y`

Keywords —

Components

Literals — `4, 11.8`

Operators — `+`, `*`

Variables — `x`, `y`

Keywords — `import`, `for`

Components

Literals — `4, 11.8`

Operators — `+`, `*`

Variables — `x`, `y`

Keywords — `import`, `for`

Expressions — `4 + x`

Statements — `y = 4 + x`

Roadmap

Week 1 (9/9) L0 Intro L1 PY - Intro	Week 2 (16/9) L2 PY – Data& Bool L3 PY – if & Loops	Week 3 (23/9) L4 PY - Lists L5 PY - Funcs	30/9 L6 PY - Files ✕
9/10 L7 PY - Mem 12/10 Exam 01	Week 6 (14/10) L8 PY - Dict L9 PY - IO	Week 7 (21/10) L10 PY – Sym I L11 PY – Sym II	Week 8 (28/10) L12 PY – Sym III L13 PY - Numpy
Week 9 (4/11) L14 PY - Plot L15 PY - State	Week 10 (11/11) L16 PY - Rand Exam 02	Week 11 (18/11) L17 PY - Debug L18 PY - Eqn	Week 12 (25/11) L19 PYA I L20 PY - Opt II
Week 13 (2/12) L21 PY - Mod L22 ML - Intro	Week 14 (9/12) L23 ML - IO L24 ML - Poly	Week 15 (16/12) L25 ML - Stats L26 ML - Fit	Week 16 (23/12) Review
Week 17 (30/12) Finals			

Objectives

A. List and distinguish each of the basic data types of Python: `int, float, complex, str`.

B. Import a *function from a library* and use it, such as `import math`.

C. Implement basic conditional logic to guide a program among various options.

D. Use attributes to expand the utility of data types.

Data Types

Numbers

$$\mathbb{N}$$

$$1, 2, 3, 4, 5, ...$$

natural numbers

Numbers

$$\mathbb{N}_0$$

$$0, 1, 2, 3, 4, 5, ...$$

whole numbers

Numbers

$$\mathbb{Z}$$

$$..., -4, -3, -2, -1, 0, +1, +2, +3, ...$$

integers

Numbers

$$\mathbb{Q}$$

$$..., -\frac{1}{4}, -\frac{1}{5}, -\frac{1}{6}, 0, +\frac{1}{3}, +\frac{2}{3}, +\frac{10}{1}, 0.25, ...$$

rational numbers

- can be expressed as a fraction by two integers.
- Is π rational?

Numbers

$$\mathbb{Q}$$

$$..., -\frac{1}{4}, -\frac{1}{5}, -\frac{1}{6}, 0, +\frac{1}{3}, +\frac{2}{3}, +\frac{10}{1}, 0.25, ...$$

rational numbers

- can be expressed as a fraction by two integers.

- Is π rational? - irrational!

Numbers

$$\mathbb{R}$$

$$\pi, e, 10^{100}, +\frac{1}{10}, 0.25, -0.11...$$

real numbers

Numbers

$$\mathbb{C}$$

Numbers

$$\mathbb{C}$$

$$i, 1 + i, \ldots$$

complex numbers
- most programming language use `j` instead of `i` for complex/imaginary number

Numbers in Python

Python supports several basic number types:

`integer`

`float`

`complex`

Numbers in Python

Python supports several basic number types:

- `integer` $\Rightarrow \mathbb{Z}$
- `float` $\Rightarrow \mathbb{R}$ or maybe $\mathbb{Q}$
- `complex` $\Rightarrow \mathbb{C}$ (again, maybe)

float , complex

Floating-point numbers include a fractional part.
(Anything with a decimal point, e.g., `2.4`, `3.0`.)

`complex` is two `float`s together.

```
0 + 1j   # ”i”
1 + 0j   # ”1”
```

How do binary numbers work?

Numeric types can be represented in binary:

`000`	0	`010`	2	`100`	4	`110`	6
`001`	1	`011`	3	`101`	5	`111`	7

Basically, in decimal:

$$513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$$

How do binary numbers work?

Numeric types can be represented in binary:

000	0	010	2	100	4	110	6
001	1	011	3	101	5	111	7

Basically, in decimal:

$$513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$$

Similarly, in binary:

$$1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

How do binary numbers work?

Numeric types can be represented in binary:

`000`	0	`010`	2	`100`	4	`110`	6
`001`	1	`011`	3	`101`	5	`111`	7

Basically, in decimal:

$$513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$$

Similarly, in binary:

$$1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$= 8 + 0 + 2 + 1 = 11$$

How do binary numbers work?

But there are only so many bytes, so there is a limit!

If we add too much, the number may **overflow**.

> 11001100 + 11000000 = ?

How do binary numbers work?

But there are only so many bytes, so there is a limit!

If we add too much, the number may **overflow**.

> 11001100 + 11000000 = ?

> = (1) 10001100

How do binary numbers work?

Python integers (`int`) can be arbitrarily large.

How do binary numbers work?

Python integers (`int`) can be arbitrarily large.

```
10 ** 100
10 ** (10 ** 5)
```

How do binary numbers work?

Python integers (`int`) can be arbitrarily large.

```
10 ** 100
10 ** (10 ** 5)
```

Floating-point numbers (`float`) have limits, though.

How do binary numbers work?

Python integers (`int`) can be arbitrarily large.

```
10 ** 100
10 ** (10 ** 5)
```

Floating-point numbers (`float`) have limits, though.

```
1.0 * 10 ** 300    # okay
1.0 * 10 ** 340    # ``infinite''
1.0 * 10 ** -400   # ``zero''
```

float

Floating-point numbers include a fractional part.
(Anything with a decimal point, e.g.,`2.4`, `3.0`.)

What are the limits?

Overflow/underflow (values too big or too small)
Arbitrary precision (i.e., number of decimal places)
- (π, *e*)

What is an encoding?

```
01001000 01000101 01001100 01001100
```

What does a binary data value like the above represent?

- What does binary data represent?
- How does the processor know?

What is an encoding?

```
01001000 01000101 01001100 01001100
```

What does a binary data value like the above represent?

- What does binary data represent?
- How does the processor know?
- The **encoding** interprets the value.

What is a data type?

A **data type** defines an encoding rule.

All values have a type.

What is a data type?

A **data type** defines an encoding rule.

All values have a type.

The type defines

- how data is represented in memory.
- the allowed operations and how they work.

Operators

Integer operations

Evaluating an expression of integers will generally result in an integer answer

```
3 + 5
```

Integer operations

Evaluating an expression of integers will generally result in an integer answer

```
3 + 5
```

EXCEPTION: DIVISION!

Integer operations

Evaluating an expression of integers will generally result in an integer answer

```
3 + 5
```

EXCEPTION: DIVISION!

`3 / 4` → `0.75`

Floating-point operations

Evaluating an expression of floating-point values will result in a floating-point answer.

Floating-point operations

Evaluating an expression of floating-point values will result in a floating-point answer.

`3.0 + 5.5` → `8.5`

Floating-point operations

Evaluating an expression of floating-point values will result in a floating-point answer.

```
3.0 + 5.5 → 8.5
3.0 + 5.0 → 8.0
```

Floating-point operations

Evaluating an expression of floating-point values will result in a floating-point answer.

`3.0 + 5.5` → `8.5`

`3.0 + 5.0` → `8.0`

`3 + 5.5` → `?` (what happens here?)

Floating-point operations

Evaluating an expression of floating-point values will result in a floating-point answer.

```
3.0 + 5.5 → 8.5
3.0 + 5.0 → 8.0
3 + 5.5 → ?
```
(what happens here?)

Engineers and scientists need to think carefully about the precision of answers.

Promotion

If one type is inadequate for a result, Python “promotes” the result.

```
1 / 3          # int => float
(-1) ** 0.5    # int => complex
(-1.0) ** 0.5 # float => complex
```

Question

```
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )
```

What is printed to the screen?

- A `40`
- B `40.3333`
- C `40.3333 + 1j`
- D None of the above

Question

```
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )
```

What is printed to the screen?

- A `40`
- B `40.3333`
- C `40.3333 + 1j` ⋆
- D None of the above

Question

```
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )
```

What is printed to the screen?

- A `40`
- B `40.3333`
- C `40.3333 + 1j` ⋆ → 2 parts → real and imaginary
- D None of the above

Attribute operator .

Attribute operator `.`.

Reaches inside of a value to access part of its data (called an attribute).

Extracts special variables stored “inside” of the type.

```
print(x.real)
print(x.imag)
```

Attribute operator .

Attribute operator `.`.

Reaches inside of a value to access part of its data (called an attribute).

Extracts special variables stored “inside” of the type.

```
print(x.real)
print(x.imag)
```

Both of these components are `float`s.

Question

```
x = (3.5 + 1j)
y = 1
z = x + y
```

What is the type of `z.imag`?

A `int`

B `float`

C `complex`

Question

```
x = (3.5 + 1j)
y = 1
z = x + y
```

What is the type of `z.imag`?

A `int`

B `float` ⋆z is `complex`, not its components!

C `complex`

Question

```
x = (3.5 + 1j)
y = 1
z = x + y
```

What is the value of `z.imag`?

A `4.5 + 1j`

B `4.5`

C `1j`

D `1.0`

Question

```
x = (3.5 + 1j)
y = 1
z = x + y
```

What is the value of `z.imag`?

A `4.5 + 1j`

B `4.5`

C `1j`

D `1.0` ⋆

Library

Library

Python offers many libraries to support other operations.

```
import math

math.factorial( 5 )
math.log( 10 )
math.pi
math.e
```

Library

Python offers many libraries to support other operations.

```
import math

math.factorial( 5 )
math.log( 10 )
math.pi
math.e
```

Note that you need to include the library name and the attribute operator `.`.

Library

Alternatively, you can retrieve one thing (name) from a library:

```
from math import log
log( 10 )

from math import factorial
factorial( 5 )

from math import pi
```

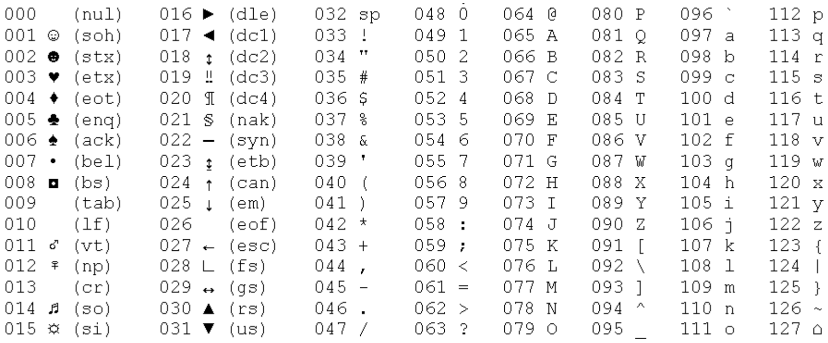
String Data Type

How does text work?

Each symbol is stored individually, one byte long:

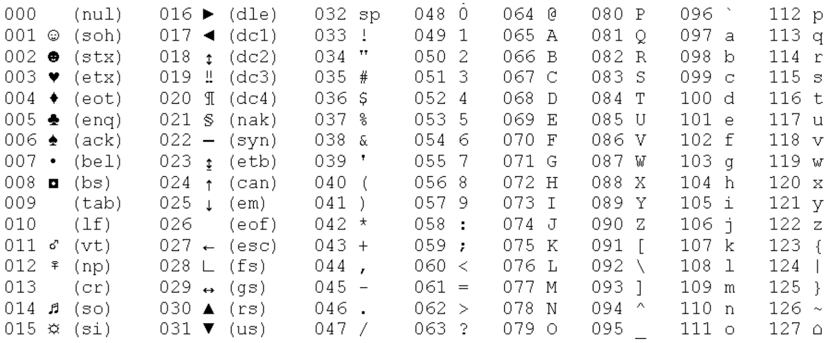
01001000	72
01000101	69
01001100	76
01001100	76
01001111	79

ASCII encoding table



000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

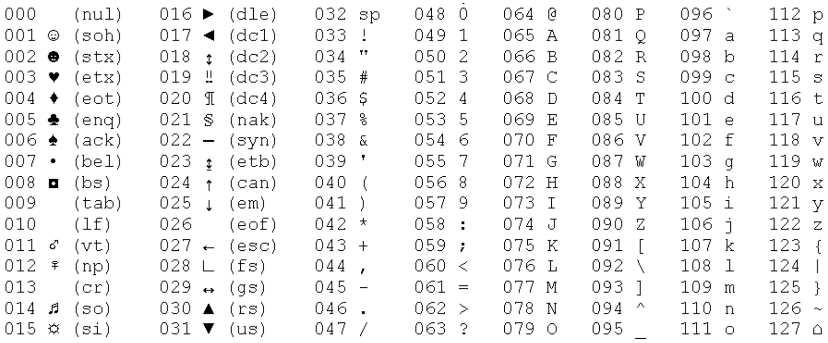
ASCII encoding table



000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

72 69 76 76 79 =

ASCII encoding table



000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

72 69 76 76 79 = H E L L O

Strings

As a literal: text surrounded by quotes (single or double).

```
'DEEP'
"DEEP"
```

Strings

As a literal: text surrounded by quotes (single or double).

′DEEP′

″DEEP″

”’DEEP”’ ?

Strings

As a literal: text surrounded by quotes (single or double).

```
’DEEP’
”DEEP”
```

”’DEEP”’ ?

Each letter is a character.

Strings

As a literal: text surrounded by quotes (single or double).

```
'DEEP'
"DEEP"
```

”’DEEP”’ ?

Each letter is a character.

Unlike numeric types, strings vary in length.

String operations

Concatenation: combine two strings

Uses the `+` symbol

```
'RACE' + 'CAR'
```

String operations

Concatenation: combine two strings

Uses the `+` symbol

`'RACE' + 'CAR'`

Repetition: repeat a string

Uses the `*`

`'HELLO '*10`

String operations

Concatenation: combine two strings

Uses the `+` symbol

`’RACE’ + ’CAR’`

Repetition: repeat a string

Uses the `*`

`’HELLO ’*10`

Formatting: encode other data type as string

Uses the `%` symbol

Formatting operator %

Creates a string

Replaces unknown with a value

- Formats nicely
- Requires indicator of type inside of string

Formatting operator %

Creates a string

Replaces unknown with a value

- Formats nicely
- Requires indicator of type inside of string

```
x = 100 * 54
s = ”String is: %i” % x
print(s)

’String is: 5400’
```

Formatting Print

You can also format your output (here, in Python 2 and C style). Limited and does not store the variable.

```
x = 65,

print(’%d’ %x) = ‘65‘
- (’%i’, ’%d’ are valid for integers)

print(’%f’ %x) = ’65.000000‘
- (’%.2f’ gives an output with 2 dec. places)

print(’%c’ %x) = ’A’ (why?)
- (’%s’ returns a string)
```

Formatting Print

You can also format your output (here, in Python 2 and C style). Limited and does not store the variable.

```
x = 65,

print(’%d’ %x) = ‘65‘
- (’%i’, ’%d’ are valid for integers)

print(’%f’ %x) = ’65.000000‘
- (’%.2f’ gives an output with 2 dec. places)

print(’%c’ %x) = ’A’ (why?)
- (’%s’ returns a string)
```

ans: From ASCII table

Indexing operator

Extracts single character or a range of characters

```
a = ”FIRE”
a[0]
```

Indexing operator

Extracts single character or a range of characters

```
a = ”FIRE”
a[0]
```

The integer is the *index*.

We count from zero!

If *negative*, counts down from end.

Slicing operator :

Extracts range of characters (*substring*)

Range specified inside of indexing operator

```
a = ”FIREHOUSE”
a[0:4]
```

Slicing operator :

Extracts range of characters (*substring*)

Range specified inside of indexing operator

```
a = ”FIREHOUSE”
a[0:4]
```

Can be a bit tricky at first:

- **Includes** character at **first index**
- **Excludes** character at **last index**

Example

```
alpha = ”ABCDE”
x = alpha[1:3]
```

What is the value of `x`?

A `’AB’`

B `’ABC’`

C `’BC’`

D `’BCD’`

E `’CD’`

Example

```
alpha = ”ABCDE”
x = alpha[1:3]
```

What is the value of `x`?

A ’AB’

B ’ABC’

C ’BC’ ⋆

D ’BCD’

E ’CD’

Type Conversion

You can convert one data type to another in Python using:

x = 12 => (this is an `int`)

y = str(x) = ’12’ => (this is a `string`)

z = float(x) = 12.0 => (this is a `float`)

Note: Not all data types can be inter-converted.

User Input

User input

`input` is a built-in function.

Argument: string prompting user

Return value: input from user (as `str`!)

```
a = input(”Enter a number:”)
```

- `a` is of string type

Boolean Logic

Boolean

`bool` is a type with two possible values:

```
    True
    False
```

We use these to make decisions.

The logic is based on *Boolean algebra*.

Boolean

`bool` is a type with two possible values:

```
True
False
```

We use these to make decisions.

The logic is based on *Boolean algebra*.

Operators:

```
and
or
not
```

Boolean operators

Operators:

- `and` : `True` only if both sides are `True`
- `or` : `True` if either side is `True`
- `not` : swaps `False` and `True`

Question

```
x = (True and False) and not (True or False)
```

What is the value of `x`?

A `True`

B `False`

C `Confused?`

Question

```
x = (True and False) and not (True or False)
```

What is the value of `x`?

```
x = (False) and not (True)
```

Question

```
x = (True and False) and not (True or False)
```

What is the value of `x`?

```
x = (False) and not (True)
```

```
x = (False) and (False)
```

Question

```
x = (True and False) and not (True or False)
```

What is the value of `x`?

A `True`

B `False` ⋆

Comparison operators

These produce Boolean output.

- less than, <
- greater than, >
- less than or equal to, <=
- greater than or equal to, >=
- equal to, ==
- not equal to, !=

Fun time

```
a = ’ZJUI’
b = ’UIUC’

x = a < b and a[1] != b[-2]
```

What is the value of `x`?

A `True`

B `False`

Solution

```
a = 'ZJUI'
b = 'UIUC'

x = a < b and a[1] != b[-2]
```

What is the value of `x`?

A `True`

B `False` ⋆

Summary

Summary

1. `int`, `float`, `complex`
2. `str`, operators + *, slice using [:].
3. ASCII table
4. attributes like `.real`
5. `input( )`
6. `bool`
7. `import` libraries