

Python Basics!

data types, strings, indexing

CS101 Lecture #3

Administrivia

- Homework #2–#6 will be later.

Administrivia

- Homework #2–#6 will be later.
- Final answer counts.

Administrivia

- Homework #2–#6 will be later.
- Final answer counts.
- Answers will be released 18 hours later.

- ✦ Lab #2 tomorrow Sunday.

- ❖ Lab #2 tomorrow Sunday.
- ❖ Where can you get help in this class?
 - ❑ Blackboard forum
 - ❑ Instructors in labs and office hours

- ❖ Lab #2 tomorrow Sunday.
- ❖ Where can you get help in this class?
 - ❑ Blackboard forum
 - ❑ Instructors in labs and office hours
- ❖ You don't need to install Python—but if you do, use Python 3.

- ❖ Lab #2 tomorrow Sunday.
- ❖ Where can you get help in this class?
 - ❑ Blackboard forum
 - ❑ Instructors in labs and office hours
- ❖ You don't need to install Python—but if you do, use Python 3.
- ❖ This is not a “weeder” class—you can succeed!

Quick Review & A Bit New

How Assignment Works

`x = 10`

How Assignment Works

```
x = 10  
y = x * x
```

How Assignment Works

```
x = 10  
y = x * x  
x * x = y
```

How Assignment Works

```
x = 10  
y = x * x  
x * x = y
```

```
x,y = y,x # a neat trick
```

Warmup Quiz

Question #1

```
x = 10  
y = x + 1  
y = x * y
```

What is the value of y?

- A 11
- B 100
- C 110
- D None of the above

Question #2

```
x = 10  
y = x + 1  
y = x * y
```

What do we call `x`?

- A a literal
- B a variable
- C an expression
- D a statement

Question #3

```
x = 10  
y = x + 1  
y = x * y
```

What do we call 10?

- A a literal
- B a variable
- C an expression
- D a statement

Question #2

```
x = 10  
y = x + 1  
y = x * y
```

What do we call $y = x * y$?

- A a literal
- B a variable
- C an expression
- D a statement

Question #5

$$x = 10$$

$$y = x$$

$$x = 5$$

What is the value of y ?

A 10

B 5

Reminder

- ▣ You will have graded quiz starting from the upcoming Monday lecture!

Data Types

What is an encoding?

01001000 01000101 01001100 01001100

What does a binary data value like this represent?

- ❖ What does binary data represent?

*What is an **encoding**?*

01001000 01000101 01001100 01001100

What does a binary data value like this represent?

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

*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.

*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 type defines allowed operations and how they work.

Example

01100111 can be the number 103, the letter g, hexadecimal 67, etc.

- ✦ So what are these data types?

Numeric Data Types

Representing numbers in binary

- Binary can naturally represent 'natural numbers' :

00000000	0	00000100	4	00001000	8
00000001	1	00000101	5	00001001	9
00000010	2	00000110	6	...	
00000011	3	00000111	7	11111111	255

Representing numbers in binary

- Binary can naturally represent 'natural numbers' :

00000000	0	00000100	4	00001000	8
00000001	1	00000101	5	00001001	9
00000010	2	00000110	6	...	
00000011	3	00000111	7	11111111	255

https://en.wikipedia.org/wiki/Binary_number
<https://www.bottomupcs.com/chapter01.xhtml#d0e660>

Integers (int), \mathbb{Z}

- ❖ How about *Integers*?

..., -3, -2, -1, 0, 1, 2, 3, ...

- ❖ Negative numbers?

Integers (int), \mathbb{Z}

- ❖ How about *Integers*?

..., -3, -2, -1, 0, 1, 2, 3, ...

- ❖ Negative numbers?
- ❖ Use the leftmost bit as **sign bit**.

Integers (*int*), \mathbb{Z}

- ❖ How about *Integers*?

..., -3, -2, -1, 0, 1, 2, 3, ...

- ❖ Negative numbers?
- ❖ Use the leftmost bit as **sign bit**.
- ❖ What are the limits of a 8-bit integer representation (with negatives)?

Integers (*int*), \mathbb{Z}

- ❖ How about *Integers*?

..., -3, -2, -1, 0, 1, 2, 3, ...

- ❖ Negative numbers?
- ❖ Use the leftmost bit as **sign bit**.
- ❖ What are the limits of a 8-bit integer representation (with negatives)?

-128...127

History

- ❖ Old version python **int** are 32 bits long (in the range of -2^{31} to $2^{31} - 1$)
- ❖ That's -2147483648 to 2147483647
- ❖ values too big: *overflow*
- ❖ values too small: *underflow*

- ❖ Python has another integer type: **long**
- ❖ Represents with no restrictions on size (no overflow/underflow)

- ❖ Python has another integer type: **long**
- ❖ Represents with no restrictions on size (no overflow/underflow)
- ❖ Since v2.2, python converts int overflow to a **long**

- ❖ Python has another integer type: **long**
- ❖ Represents with no restrictions on size (no overflow/underflow)
- ❖ Since v2.2, python converts int overflow to a **long**
- ❖ newer Python versions promises there is no distinction between **int** and **long**

History

- ❖ Python has another integer type: **long**
- ❖ Represents with no restrictions on size (no overflow/underflow)
- ❖ Since v2.2, python converts int overflow to a **long**
- ❖ newer Python versions promises there is no distinction between **int** and **long**
- ❖ Don't get spoiled by this (many languages still have clear integer types and limits).

[https://en.wikipedia.org/wiki/Integer_\(computer_science\)](https://en.wikipedia.org/wiki/Integer_(computer_science))

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!**
 - ❑ $3 / 4 \rightarrow 0.75$

Integer operations

- ❖ Evaluating an expression of integers will generally result in an integer answer
 - ❑ $3 + 5$
 - ❑ **EXCEPTION: DIVISION!**
 - ❑ $3 / 4 \rightarrow 0.75$
 - ❑ $3 // 4 \rightarrow 0$ (floor division)

Integer operations

- ❖ Evaluating an expression of integers will generally result in an integer answer
 - $3 + 5$
 - **EXCEPTION: DIVISION!**
 - $3 / 4 \rightarrow 0.75$
 - $3 // 4 \rightarrow 0$ (floor division)
 - $4 / 2 \rightarrow ??$

Floating-point numbers, \mathbb{R}

- ▣ Floating-point numbers include a fractional part.

Floating-point numbers, \mathbb{R}

- ❖ Floating-point numbers include a fractional part.
(Anything with a decimal point—2.4, 3.0.)

Floating-point numbers, \mathbb{R}

- ❖ Floating-point numbers include a fractional part.
(Anything with a decimal point—2.4, 3.0.)
- ❖ What are limits?

Floating-point numbers, \mathbb{R}

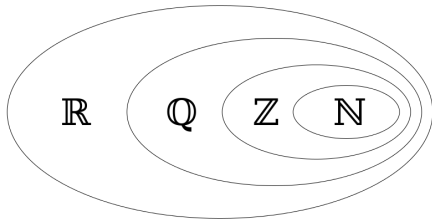
- ❖ Floating-point numbers include a fractional part.
(Anything with a decimal point—2.4, 3.0.)
- ❖ What are limits?
 - ❑ Overflow/underflow
 - ❑ Precision (π , e)

Floating-point numbers, \mathbb{R}

- ❖ Floating-point numbers include a fractional part.
(Anything with a decimal point—2.4, 3.0.)
- ❖ What are limits?
 - ❑ Overflow/underflow
 - ❑ Precision (π , e)
- ❖ Float representation in Binary

<http://sites.cs.queensu.ca/courses/cisc121/Record/Week09/Arith.pdf>

Floating-point numbers, \mathbb{R}



Real numbers (\mathbb{R}) include the rational (\mathbb{Q}), which include the integers (\mathbb{Z}), which include the natural numbers (\mathbb{N}).

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.
 - $3.0 + 5.5 \rightarrow 8.5$

Floating-point operations

- ❖ Evaluating an expression of floating-point values will result in a floating-point answer.
 - ❑ $3.0 + 5.5 \rightarrow 8.5$
 - ❑ $3.0 + 5.0 \rightarrow 8.0$

Floating-point operations

- ❖ Evaluating an expression of floating-point values will result in a floating-point answer.
 - ❑ $3.0 + 5.5 \rightarrow 8.5$
 - ❑ $3.0 + 5.0 \rightarrow 8.0$
 - ❑ $3 + 5.5 \rightarrow ?$ (what happens here?)

Floating-point operations

- ❖ Evaluating an expression of floating-point values will result in a floating-point answer.
 - $3.0 + 5.5 \rightarrow 8.5$
 - $3.0 + 5.0 \rightarrow 8.0$
 - $3 + 5.5 \rightarrow ?$ (what happens here?)
- ❖ Engineers and scientists need to think carefully about data type, precision, and the *conversion*.

Complex numbers, \mathbb{C}

- ✦ Represent numbers with an imaginary component.

Complex numbers, \mathbb{C}

- ❖ Represent numbers with an imaginary component.
- ❖ Use j for i :

$$1.0 + 1j$$

$$2 + 0j$$

Example

```
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

Attribute operator .

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

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 .

- ❖ 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 floats.

Example

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

What is the value of `z.imag`?

Example

```
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`

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	L (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	ℳ (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	L (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

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	L (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
'HELLO'

- As a literal: text surrounded by quotes.
 - "DEEP"

Strings

- ❖ As a literal: text surrounded by quotes.
 - ❑ "DEEP"
- ❖ Each symbol is a character.

Strings

- ❖ As a literal: text surrounded by quotes.
 - ❑ "DEEP"
- ❖ Each symbol 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:** used to encode other data as string
 - Uses % symbol

Formatting operator

- ✦ Creates string with value inserted

Formatting operator

- ❖ Creates string with value inserted
 - Formats nicely
 - Requires indicator of type inside of string

Formatting operator

- ❖ Creates string with value inserted
 - Formats nicely
 - Requires indicator of type inside of string

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

Example

```
name = "Tao"  
grade = 2 / 3  
m1 = "Hello, %s!" % name  
m2 = "Your grade is: %f." % grade  
print(m1)  
print(m2)
```

Example

```
name = "Tao"  
grade = 2 / 3  
m1 = "Hello, %s!" % name  
m2 = "Your grade is: %f." % grade  
print(m1)  
print(m2)
```

```
Hello, Tao!  
Your grade is 0.66667.
```


Example

```
x = 3
s = ("%i" % (x+1)) * x**(5%x)
print(s)
```

What does this program print?

A 3333333333333

B 4444444444

C 9999

D %i%i%i%i%i

Indexing operator

- ✦ Extracts single character

Indexing operator

- ✦ Extracts single character
a = "FIRE"
a[0]

Indexing operator

- ❖ Extracts single character
a = "FIRE"
a[0]
- ❖ The integer is the index.

Indexing operator

- ❖ Extracts single character
a = "FIRE"
a[0]
- ❖ The integer is the index.
- ❖ **We count from zero!** (same in C, C++, Java)

Indexing operator

- ❖ Extracts single character
a = "FIRE"
a[0]
- ❖ The integer is the index.
- ❖ **We count from zero!** (same in C, C++, Java)
- ❖ If negative, counts down from end.
- ❖ a[-1] refers to the last character

Question

```
s = "ABCDE"  
i = 3  
x = s[i]
```

What is the value of x?

- A 'A'
- B 'B'
- C 'C'
- D 'D'
- E 'E'

Question

```
s = "ABCDE"  
i = 25 % 3  
y = s[i]
```

What is the value of y?

- A 'A'
- B 'B'
- C 'C'
- D 'D'
- E 'E'

Question

```
s = "ABCDE"  
i = (11 % 3) - 7  
z = s[i]
```

What is the value of z?

- A 'A'
- B 'B'
- C 'C'
- D 'D'
- E 'E'

Question

```
s = "ABCDE"  
i = (11 % 3) - 7  
z = s[i]
```

What is the value of z?

- A 'A'
- B 'B'
- C 'C'
- D 'D'
- E 'E'

Question

```
s = "ABCDE"  
i = (11 % 3) + 3  
z = s[i]
```

What is the value of z?

Question

```
s = "ABCDE"  
i = (11 % 3) + 3  
z = s[i]
```

What is the value of `z`?
How about `s[-6]`?

Reminders

Reminders

- ❖ Lab #2 tomorrow Sunday.