Python Basics!

data types, strings, indexing

CS101 Lecture #3

2016-08-29

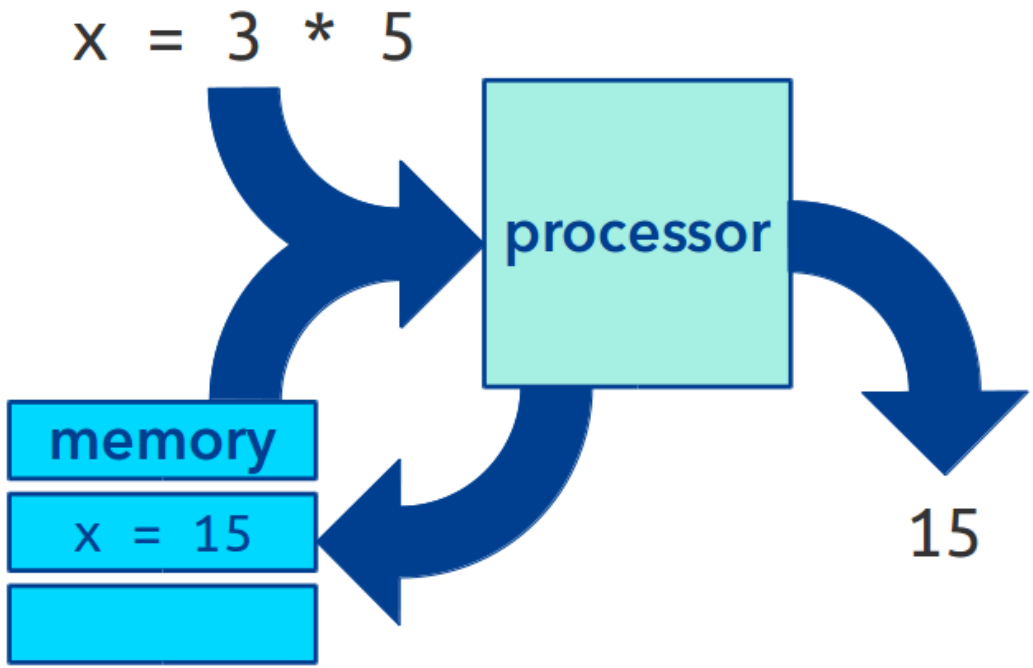
Administrivia

Administrivia

- Register your i>clickers on the course Compass page.
- Complete Homework #1 before Wednesday at 5:00 p.m.
- Lab #2 this week, no lab next week (Labor Day).

Warmup Quiz

Our execution model



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

Data Types

What is an **encoding**?

```
01001000 01000101 01001100 01001100
```

What does this binary data value represent?

- What does this binary data represent?

What is an **encoding**?

`01001000 01000101 01001100 01001100`

What does this binary data value represent?

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

What is an **encoding**?

`01001000 01000101 01001100 01001100`

What does this binary data value represent?

- What does this 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.

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`, `3.5`, etc.

- So what are these data types?

Numeric Data Types

How do binary numbers work?

- Numeric types can be represented in binary:

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

How do binary numbers work?

- Numeric types can be represented in binary:

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

- If we add more, the number **overflows**.

How do binary numbers work?

- Numeric types can be represented in binary:

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

- If we add more, the number **overflows**.
- Negative numbers? Add a **sign bit**.

Integers, $\mathbb{Z}$

- Integers have been our only type thus far.

$$\ldots, -4, -3, -2, -1, 0, +1, +2, +3, \ldots$$

- What are limits?

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`
  - `3 // 4` → `0` (floor division)

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
  - Arbitrary precision (π,)

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`
  - `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 the precision of answers.

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
```

Complex numbers, $\mathbb{C}$

- Represent numbers with an imaginary component.
- Use `j` for i:
  ```
  1.0 + 1j
  ```
- Think of "jmaginary" numbers, I suppose.

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” 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" 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`?

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

	`01001000`	72
	`01000101`	69
long:	`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 ’HELLO’

Strings

- 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 = ”Neal”
grade = 2 / 3
m1 = ”Hello, %s!”” % name
m2 = ”Your grade is:  %f” % grade
print(m1)
print(m2)
```

Example

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

What does this program print?

A `333333333333`
B `444444444`
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!

Indexing operator

- Extracts single character

```
a = ”FIRE”
a[0]
```

- The integer is the index.
- We count from zero!
- If negative, counts down from end.

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’

Reminders

Reminders

- Register your i>clicker on Compass.
  (last chance before it counts!)
- Homework #1 due Wednesday, Aug. 31, 5:00 p.m.