Python Basics!

strings, functions, scope

CS101 Lecture #4

2017-9-28

Administrivia

Administrivia

- Homework #2 is due Tuesday Oct. 3.

Data Types—Strings

ASCII 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 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	⌂

The table provides an *encoding* scheme from symbols to numbers

72 69 76 76 79 = H E L L O

How to store text on computer?

- `H E L L O` = 72 69 76 76 79

How to store text on computer?

- `H E L L O` = 72 69 76 76 79
- Each symbol is stored individually, one byte long:

How to store text on computer?

- `H E L L O` = 72 69 76 76 79
- Each symbol is stored individually, one byte long:

72	01001000
69	01000101
76	01001100
76	01001100
79	01001111

How to store text on computer?

- `H E L L O` = 72 69 76 76 79
- Each symbol is stored individually, one byte long:

72	01001000
69	01000101
76	01001100
76	01001100
79	01001111

`'HELLO'` : 01001000 01000101 01001100 01001100 01001111

Strings

Strings

- As a literal: text surrounded by quotes.
  - `'DEEP'`, or `"DEEP"`

Strings

- As a literal: text surrounded by quotes.
  - `'DEEP'`, or `"DEEP"`
  - single quote(`''`) and double quote (`""`) are equivalent in python (not in C or C++)

Strings

- As a literal: text surrounded by quotes.
  - `'DEEP'`, or `"DEEP"`
  - single quote(`''`) and double quote (`""`) are equivalent in python (not in C or C++)
- Can have arbitrary length, including an empty string (`''`).

Strings

- As a literal: text surrounded by quotes.
  - `'DEEP'`, or `"DEEP"`
  - single quote(`''`) and double quote (`""`) are equivalent in python (not in C or C++)
- Can have arbitrary length, including an empty string (`''`).
- Each element is a character — also a string type

Strings

- As a literal: text surrounded by quotes.
  - `'DEEP'`, or `"DEEP"`
  - single quote(`''`) and double quote (`""`) are equivalent in python (not in C or C++)
- Can have arbitrary length, including an empty string (`''`).
- Each element is a character — also a string type
- In C/C++, a character is a different data type (`char`)

Strings

```
'the quick brown fox jumps over a lazy dog'
```

String operations

- **Concatenation**: combine two strings
  - Uses the + symbol (operator for string concatenation)

String operations

- **Concatenation**: combine two strings
  - Uses the + symbol (operator for string concatenation)
  - `'RACE' + 'CAR'`

String operations

- **Concatenation**: combine two strings
  - Uses the + symbol (operator for string concatenation)
  - `'RACE' + 'CAR'`
  - the "same" operator works differently with different types of operand (*operator overload*)

String operations

- **Concatenation**: combine two strings
  - Uses the + symbol (operator for string concatenation)
  - `'RACE' + 'CAR'`
  - the "same" operator works differently with different types of operand (*operator overload*)

```
1 + 2 = 3
'RACE' + 'CAR' = 'RACECAR'
```

String operations

- **Repetition**: repeat a string
  - Uses the *
  - `'HELLO '*10`

String operations

- **Formatting**: Creates string with other data types inserted in

String operations

- **Formatting**: Creates string with other data types inserted in
  - Requires a special indicator of Formatting: %

String operations

- **Formatting**: Creates string with other data types inserted in
  - Requires a special indicator of Formatting: %
  - Requires indicator of data type

String operations

- **Formatting**: Creates string with other data types inserted in
  - Requires a special indicator of Formatting: %
  - Requires indicator of data type

```
x = 100 * 54
s = "The value of x is: %i" % x
print(s)
```

String operations

- **Formatting**: Creates string with other data types inserted in
  - Requires a special indicator of Formatting: %
  - Requires indicator of data type

```
x = 100 * 54
s = "The value of x is: %i" % x
print(s)
```

```
The value of x is: 5400
```

Formatting operator

- Creates string with value inserted
  - Indicators for different data types

```
'%i'  int
'%f'  float
'%e'  float (scientific notation)
'%s'  str
```

Example

```
print( 'An integer:   %i' % 7 )
print( 'A float:     %f' % 7.0 )
print( 'A float:     %e' % 7.0 )
print( 'A string:    %s' % 'seven' )
```

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 0.666667.
```

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

Escape character ' \ '

- Read as "back slash"
- Defines special characters
  - `\n` new line
  - `\t` tab (tabular key)
  - `\v` vertical tab

Escape character ' \ '

- Read as "back slash"
- Defines special characters
  - `\n` new line
  - `\t` tab (tabular key)
  - `\v` vertical tab
- De-specialize special characters

Escape character ' \ '

- Read as "back slash"
- Defines special characters
  - `\n` new line
  - `\t` tab (tabular key)
  - `\v` vertical tab
- De-specialize special characters
  - `\\` \
  - `\'` '
  - `\"` "

Escape character ' \ '

- Read as "back slash"
- Defines special characters
  - `\n` new line
  - `\t` tab (tabular key)
  - `\v` vertical tab
- De-specialize special characters
  - `\\` \
  - `\'` '
  - `\"` "

https://docs.python.org/2.0/ref/strings.html

Example

Try:

```
print('Hello, Tao!\nYour grade is 0.667')
print('3 plus 5 is:\t%i', % (3+5))
print('put .ipynb file in C:\\Users\\nick')
```

```
Hello, Tao!
Your grade is 0.667
```

Example

Try:

```
print('Hello, Tao!\nYour grade is 0.667')
print('3 plus 5 is:\t%i', % (3+5))
print('put .ipynb file in C:\\Users\\nick')
```

```
Hello, Tao!
Your grade is 0.667
3 plus 5 is:   8
```

Example

Try:

```
print('Hello, Tao!\nYour grade is 0.667')
print('3 plus 5 is:\t%i', % (3+5))
print('put .ipynb file in C:\\Users\\nick')
```

```
Hello, Tao!
Your grade is 0.667
3 plus 5 is:   8
put .ipynb file in C:\Users\nick
```

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.

Indexing operator []

- Extracts single character
  ```
  a = "FIRE"
  a[0]
  ```
- The integer is the index.
- We count from zero!
- If negative, counts down from end.
- `a[-1]` is the last element

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'`

Slicing operator :

- Extracts a range of characters (*substring*)
- Index range specified inside `[]`

```
a = "FIREHOUSE"
a[0:4]
```

Slicing operator :

- Extracts a range of characters (*substring*)
- Index range specified inside `[]`

```
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'`

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]
```

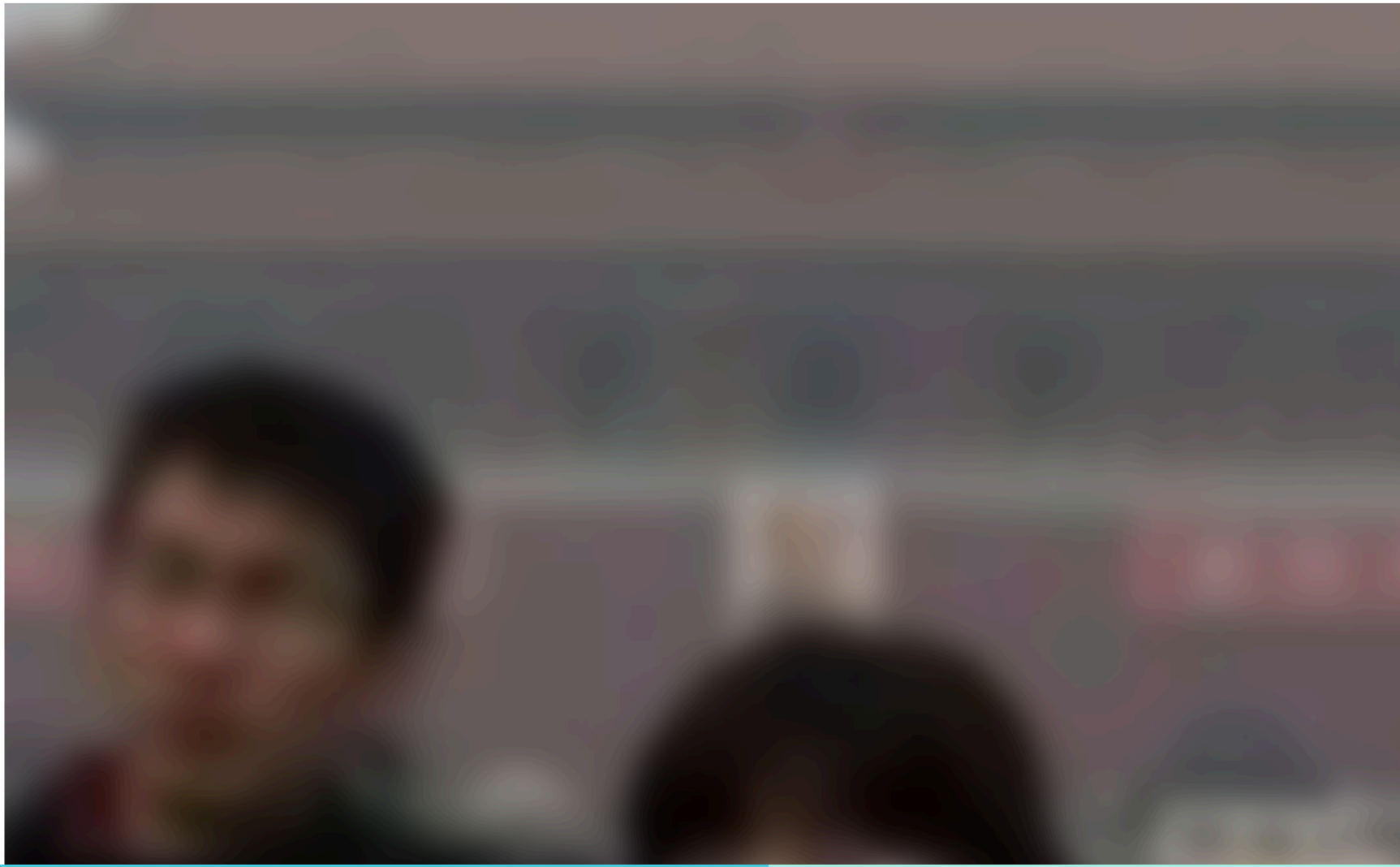
What is the value of z?

Error: Out-of-Index!

Secure https://weather.com/news/news/air-pollution-china-beijing-off-index-20130113

Air Pollution in Beijing Goes Off the Index

By Louise Watt · January 14 2013 12:07 PM EDT · Associated Press



Functions

And now for something different...

- A *function* is a piece of code (code block) we can execute with a single line.

And now for something different...

- A *function* is a piece of code (code block) we can execute with a single line.
  - Provides an interface that *encapsulates* a series of actions

And now for something different...

- A *function* is a piece of code (code block) we can execute with a single line.
  - Provides an interface that *encapsulates* a series of actions
  - Saves us from rewriting code

And now for something different...

- A *function* is a piece of code (code block) we can execute with a single line.
  - Provides an interface that *encapsulates* a series of actions
  - Saves us from rewriting code
  - Makes code cleaner and easy to read

And now for something different...

- A *function* is a piece of code (code block) we can execute with a single line.
  - Provides an interface that *encapsulates* a series of actions
  - Saves us from rewriting code
  - Makes code cleaner and easy to read
  - Serves as building blocks to build up bigger programs

And now for something different...

- A *function* is a piece of code (code block) we can execute with a single line.
  - Provides an interface that *encapsulates* a series of actions
  - Saves us from rewriting code
  - Makes code cleaner and easy to read
  - Serves as building blocks to build up bigger programs
- Analogy: Functions are like verbs.

Function calls

- Use name of the function with parentheses.
  - `print()`

Function calls

- Use name of the function with parentheses.
  - `print()`
- Many functions come built-in to Python or in the standard library.

Function calls

- Use name of the function with parentheses.
  - `print()`
- Many functions come built-in to Python or in the standard library.
- Others we will compose at need.

Arguments

- Functions can take data as input.

Arguments

- Functions can take data as input.
- The data that passes to a function is called *Arguments*.

Arguments

- Functions can take data as input.
- The data that passes to a function is called *Arguments*.
  - `print(5)`

Arguments

- Functions can take data as input.
- The data that passes to a function is called *Arguments*.
  - `print(5)`
  - `len('Rex Kwon Do')`

Arguments

- Functions can take data as input.
- The data that passes to a function is called *Arguments*.
  - `print(5)`
  - `len('Rex Kwon Do')`
  - `abs(-123)`

Arguments

- Functions can take data as input.
- The data that passes to a function is called *Arguments*.
  - `print(5)`
  - `len('Rex Kwon Do')`
  - `abs(-123)`
- A function can accept zero to many arguments.

Arguments

- Functions can take data as input.
- The data that passes to a function is called *Arguments*.
  - `print(5)`
  - `len('Rex Kwon Do')`
  - `abs(-123)`
- A function can accept zero to many arguments.
- Multiple arguments are separated by commas:
  - `min(1, 4, 5)`
  - `max(1, 4, 5)`

Return value

- Functions can return a result: **return value**.

Return value

- Functions can return a result: **return value**.
- Return values are the output of a function.

Return value

- Functions can return a result: **return value**.
- Return values are the output of a function.
  - `a = min(1, 4, 5)`
  - `b = max(1, 4, 5)`

Return value

- Functions can return a result: **return value**.
- Return values are the output of a function.
  - `a = min(1, 4, 5)`
  - `b = max(1, 4, 5)`
- Can return nothing
  - `print(5)`

Return value

- Functions can return a result: **return value**.
- Return values are the output of a function.
  - `a = min(1, 4, 5)`
  - `b = max(1, 4, 5)`
- Can return nothing
  - `print(5)`
  - `a = print(5)`

Return value

- Functions can return a result: **return value**.
- Return values are the output of a function.
  - `a = min(1, 4, 5)`
  - `b = max(1, 4, 5)`
- Can return nothing
  - `print(5)`
  - `a = print(5)`
  - a is a `'NoneType'`

Example

```
def f(x):
    y = x**2
    area = 0.5*math.pi * y
    return area
```

Example

```
def f(x):
    y = x**2
    area = 0.5*math.pi * y
    return area
```

Example

```
def f(x):
    y = x**2
    area = 0.5*math.pi * y
    return area
```

Example

```
def f(x):
    y = x**2
    area = 0.5*math.pi * y
    return area
```

Example

```
def f(x):
    y = x**2
    area = 0.5*math.pi * y
    return area
```

Type conversion

- A set of built-in functions to convert data from one type to another.

Type conversion

- A set of built-in functions to convert data from one type to another.
  - `float('3.14')`
  - `str(3.14)`
  - `int(3.14)`

Type conversion

- A set of built-in functions to convert data from one type to another.
  - `float('3.14')`
  - `str(3.14)`
  - `int(3.14)`
- Be careful of nonsense:
  - `int('Rex')`
  - `int(3 + 5j)`

User input

```
x = input()
```

User input

- `x = input()`
- Takes keyboard input typed by user

User input

- `x = input()`
- Takes keyboard input typed by user
- Return value: keyboard input from user (as `str`)
- No argument to the function

User input

- `x = input()`
- Takes keyboard input typed by user
- Return value: keyboard input from user (as `str`)
- No argument to the function
  - cannot skip the parentheses!

Reminders

Reminders

- Homework #2 is due Tuesday Oct. 3.

Reminders

- Homework #2 is due Tuesday Oct. 3.
- Opps, no class next week