Python Basics!

functions, scope

CS101 Lecture #4

2016-10-10

Administrivia

Administrivia

- Homework #2 is due Wed Oct. 19.

Data Types—A Few Points

Complex numbers, $\mathbb{C}$

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

```
z = 1.0 + 1j
```

Complex numbers, $\mathbb{C}$

- Represent numbers with an imaginary component.
- Use `j` for *i*:
  ```
  z = 1.0 + 1j
  ```
- `z.real + z.imag * 1j`

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 (not in C or C++)

Strings

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

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

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

String operations

- **Formatting**: used to encode other data as string
  - Uses % symbol
  - 'The revenue of this month is *100,000* HKD'

Formatting operator

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

```
'%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' )
```

Special characters

- Special characters defined in ASCII, start with \
  "\n" new line
  "\t" tab
  "\\" \
- print('the result of 3 plus 5 is:\n%i', % (3+5))
- print('the result of 3 plus 5 is:\t%i', % (3+5))
- print('put file in: C:\\ Users\\ john\\')

Quote in Quote

- “Emily’s not coming today”
  - try: print('Emily's not coming today')
  - try: print("Emily's not coming today")

Quote in Quote

- “Emily’s not coming today”
  - try: print('Emily's not coming today')
  - try: print("Emily's not coming today")
- “Movie of the week is "Kongfu Panda"”
  - try: print('Movie of the week is "Kongfu Panda"')
  - try: print("Movie of the week is "Kongfu Panda" ")

Quote in Quote

- “Emily’s not coming today”
  - try: print('Emily's not coming today')
  - try: print("Emily's not coming today")
- “Movie of the week is "Kongfu Panda"”
  - try: print('Movie of the week is "Kongfu Panda"')
  - try: print("Movie of the week is "Kongfu Panda" ")
- rule-of-thumb
  - use double quote to enclose single quote
  - use single quote to enclose double quote
  - use \' or \" for quotes inside a string

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.
- Does this work on other data types like `int`?

Slicing operator :

- Extracts range of characters (*substring*)

Slicing operator :

- Extracts range of characters (*substring*)
- Range specified inside of indexing operator

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'`

Slicing operator : with stepsize

- Extracts a sub-range of characters with a *stepsize*

Slicing operator : with stepsize

- Extracts a sub-range of characters with a *stepsize*

```
alpha = "ABCDE"
alpha[1:3:2]        #last value is stepsize
```

Slicing operator : with stepsize

- Extracts a sub-range of characters with a *stepsize*

```
alpha = "ABCDE"
alpha[1:3:2]        #last value is stepsize
'B'
```

Slicing operator : with stepsize

- Extracts a sub-range of characters with a *stepsize*

```
alpha = "ABCDE"
alpha[1:3:2]        #last value is stepsize
'B'
```

- Stepsize can be negative

```
alpha[4:1:-1]
```

Slicing operator : with stepsize

- Extracts a sub-range of characters with a *stepsize*

```
alpha = "ABCDE"
alpha[1:3:2]        #last value is stepsize
'B'
```

- Stepsize can be negative

```
alpha[4:1:-1]
'EDC'
```

Slicing operator : with stepsize

- Extracts a sub-range of characters with a *stepsize*

```
alpha = "ABCDE"
alpha[1:3:2]        #last value is stepsize
'B'
```

- Stepsize can be negative

```
alpha[4:1:-1]
'EDC'
alpha[-1:1:-2]
```

Functions

And now for something different...

- A *function* is a small program (block of code) we can run within Python.

And now for something different...

- A *function* is a small program (block of code) we can run within Python.
  - Saves us from rewriting code
  - Don't reinvent the wheel!

And now for something different...

- A *function* is a small program (block of code) we can run within Python.
  - Saves us from rewriting code
  - Don't reinvent the wheel!
- Analogy: Functions are more verbs.

And now for something different...

- A *function* is a small program (block of code) we can run within Python.
  - Saves us from rewriting code
  - Don't reinvent the wheel!
- Analogy: Functions are more verbs.
- Also called subroutine or procedure.

Function calls

- When we want to execute a function, we call or invoke it.

Function calls

- When we want to execute a function, we call or invoke it.
- Use name of the function with parentheses.

Function calls

- When we want to execute a function, we call or invoke it.
- Use name of the function with parentheses.
  - `print()`

Function calls

- When we want to execute a function, we call or invoke it.
- Use name of the function with parentheses.
  - `print()`
- Many functions come built-in to Python or in the standard library.

Function calls

- When we want to execute a function, we call or invoke it.
- 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 act on data.

Arguments

- Functions can act on data.
- *Arguments* are the input to functions.

Arguments

- Functions can act on data.
- *Arguments* are the input to functions.
- Functions can return a value. (fruitful)

Arguments

- Functions can act on data.
- *Arguments* are the input to functions.
- Functions can return a value. (fruitful)
- Return values are the output of a function.

Arguments

- Functions can act on data.
- *Arguments* are the input to functions.
- Functions can return a value. (fruitful)
- Return values are the output of a function.
  - `print('10')`

Arguments

- Functions can act on data.
- *Arguments* are the input to functions.
- Functions can return a value. (fruitful)
- Return values are the output of a function.
  - `print('10')`
  - `len('Rex Kwon Do')`

Arguments

- Functions can act on data.
- *Arguments* are the input to functions.
- Functions can return a value. (fruitful)
- Return values are the output of a function.
  - `print('10')`
  - `len('Rex Kwon Do')`
  - `abs(-123)`

Arguments

- *Arguments* are values passed *to* a function.

Arguments

- *Arguments* are values passed *to* a function.
- A function can accept zero to many arguments.

Arguments

- *Arguments* are values passed *to* a function.
- A function can accept zero to many arguments.
- Multiple arguments are separated by commas:
  - `min( 1,4,5 )`
  - `max( 1,4,5 )`

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( "0.3" )`
  - `str( 3 + 5j )`

Type conversion.

- A set of built-in functions to convert data from one type to another.
  - `float( "0.3" )`
  - `str( 3 + 5j )`
- Be careful of nonsense:
  - `int( "Rex" )`
  - `int( 3 + 5j )`

User input

- `input()` is a built-in function.

User input

- `input()` is a built-in function.
- Takes keyboard input from prompt by user

User input

- `input()` is a built-in function.
- Takes keyboard input from prompt by user
- No argument to the function

User input

- `input()` is a built-in function.
- Takes keyboard input from prompt by user
- No argument to the function
- Return value: keyboard input from user (as `str`)

Goal

- A program should achieve a goal.

Goal

- A program should achieve a goal.
- Next time we will write our first nontrivial program.

Reminders

Reminders

- Homework #2 is due Wed Oct. 19.