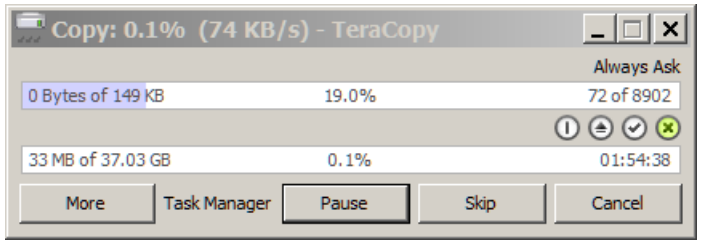
DATA 520
Lecture 9
Using Methods

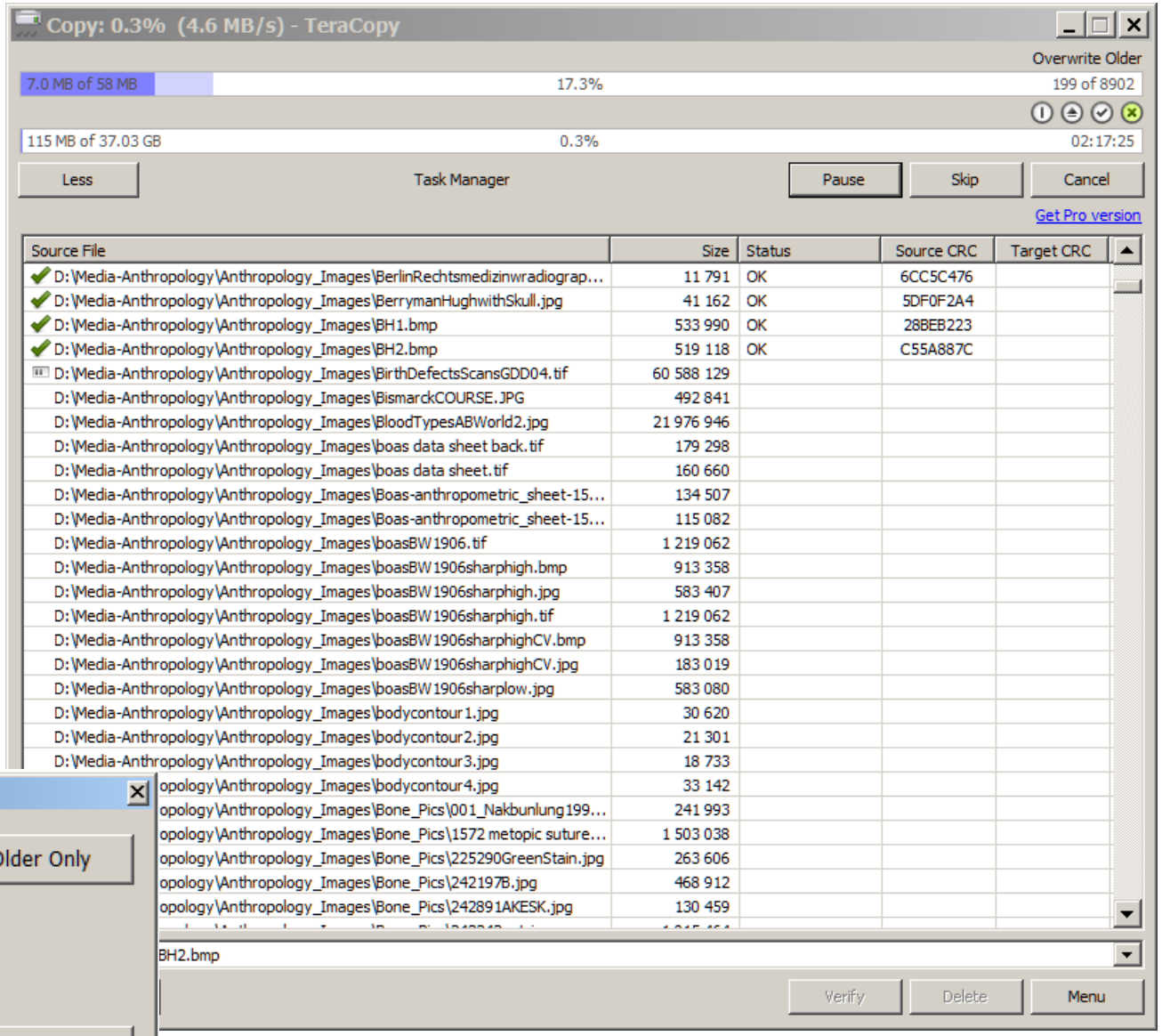
Object-oriented programming

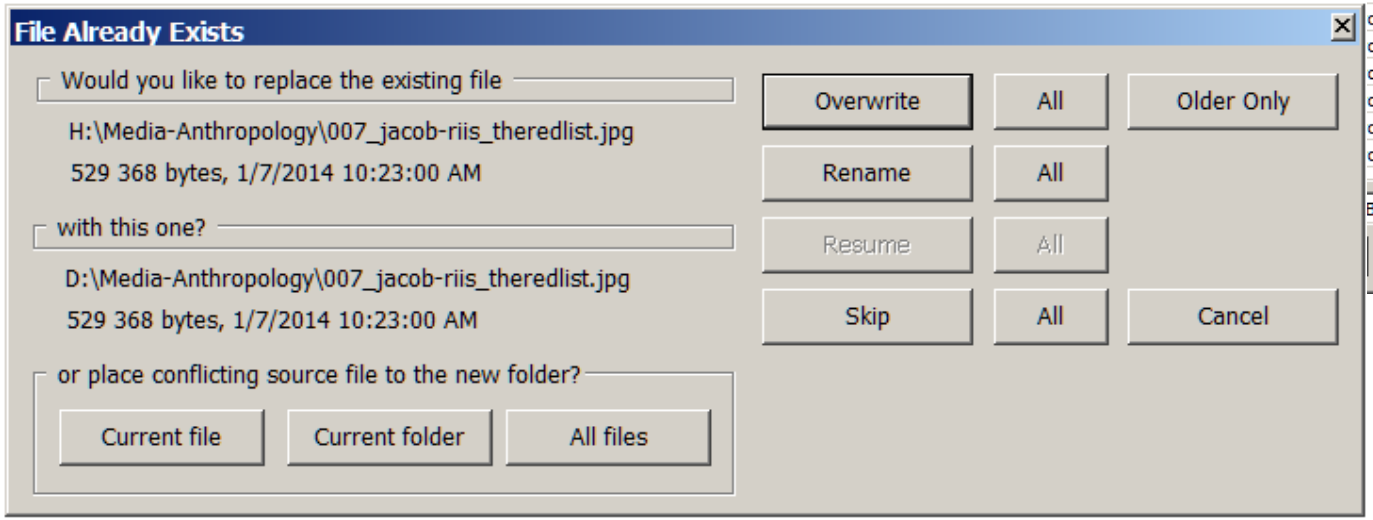
Application Monday

Teracopy

http://www.codesector.com/teracopy







Methods

A Method is like a function bound to a data item (an object of a certain type)

Strings - many many functions; can return int, Boolean, string

```
help(str) # the str module, for strings
help(str.capitalize) # one method in the str module
capitalize(...)
 S.capitalize() -> str

    Return a capitalized version of S, i.e. make the first character
    have upper case and the rest lower case.

str.capitalize('mercyhurst')
'Mercyhurst'
# shorter form - data linked to method, no "str." needed
'mercyhurst'.capitalize()
'Mercyhurst'
```

Methods

```
>>> str.lower('SIStErs oF')
'sisters of'
>>> str.upper('mercy')
'MERCY'

>>> 'Patricia'.swapcase()
'pATRICIA'
```

*** underlying string is unchanged ***

```
>>> s = 'SiSjhSjE'
>>> s.swapcase()
'sIsJHsJe'
>>> s
'SiSjhSjE'
```

Boolean result:

```
>>> str.islower('Erie')
False
>>> str.isupper('PA,USA')
True
```

Methods

Integer result: Flexible find a substring

str.find(s)

str.find(s, beg)

str.find(s, beg, end)

help(str.find)

```
Help on method_descriptor:
find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

String Methods

```
# Using str.find
'This is an example of a string.'.find('string')
24
This is an example of a string.
12345678901234567890123456
What gives?
Python starts counting with zero.
This is an example of a string.
01234567890123456789012345

'This is an example of a string.'.find('stringl')
-1
which means it was not found (NOT 0!)
'This is an example of a string.'.find('This')
0
```

String Methods

```
GriesQuote = 'This is how methods are different from functions: the first argument to every string method must be a string, and the parameter is not described in the documentation for the method. This is because all string methods require a string as the first argument, and more generally, all methods in a class require an object of that class as the first argument. Here are two more examples, this time using the other two string methods from the code on page 115. Both of these also require a string as the first argument. '

GriesQuote.find('string') # first find

78

GriesQuote.find('string',78) # still first find at 78

78

GriesQuote.find('string',79) # next (second) find (at or after 79)

102

GriesQuote.find('string',103) # next (third) find (at or after 103)

202

GriesQuote.find('string',103, 201) # really nothing between 103 and 201?

-1
```

String Methods

```
# found at 202
GriesQuote.find('string',103, 201) # really nothing between 103 and 201?
-1
GriesQuote.find('string',103, 202) # really nothing between 103 and 202?
-1
GriesQuote.find('string',103, 203) # really nothing between 103 and 203?
-1
GriesQuote.find('string',103, 204) # really nothing between 103 and 204?
-1
GriesQuote.find('string',103, 205) # really nothing between 103 and 205?
-1
GriesQuote.find('string',103, 206) # really nothing between 103 and 206?
-1
GriesQuote.find('string',103, 207) # really nothing between 103 and 207?
-1
GriesQuote.find('string',103, 208) # really nothing between 103 and 208?
202
```

String Methods

```
# found at 202
GriesQuote.find('string',103, 207) # really nothing between 103 and 207?


# get a substring
GriesQuote[103:207]
'tring, and the parameter is not described in the documentation for the method. This is because all strin'
Oh, right, 'string' was not complete!


# How many times does a substring appear in a string?
GriesQuote.count('string')
6


# str.split(s) - converts a string to a list of all words in s
str.split('The problem can be the leading and the trailing spaces.  ')
['The', 'problem', 'can', 'be', 'the', 'leading', 'and', 'the', 'trailing', 'spaces.']
```

String Methods

```
strip(...) # remove characters
 |      S.strip([chars]) -> str
 |
 |      Return a copy of the string S with leading and trailing
 |      whitespace removed.
 |      If chars is given and not None, remove characters in chars instead.
str.strip('  the problem can be leading and trailing spaces.  ')
'the problem can be leading and trailing spaces.'
# nested
str.capitalize(str.strip('  the problem can be leading and trailing spaces.  '))

str.lstrip(s) # strip from beginning (left side)
str.rstrip(s) # strip from end (right side)
```

Sometimes you want to standardize to one space before, one space after a string

- so strip first, then add a space to beginning and end

String Methods

```
# notice the parens when a string is calculated
('TTA' + 'G' * 3).strip('T')
'AGGG'
(' TTA' + 'G' * 3).strip()
'TTAGGG'

'TTAGGG'.lstrip('T')
'AGGG'

'TTAGGGTTSTTTT'.lstrip('T') # leaves those on the right
'AGGGTTSTTTT'

str.count()  # count the number of times a substring appears in a string
'How do I love thee? Let me count the ways.'.count('the')
2
('TTA' + 'G' * 3).count('T')
2
```

String Methods

```
help(str.replace)
Help on method_descriptor:

replace(...)
    S.replace(old, new[, count]) -> str

    Return a copy of S with all occurrences of substring
    old replaced by new.  If the optional argument count is
    given, only the first count occurrences are replaced.

str1 = "this is a string example....wow!!! this is really a string";
str1.replace("is", "was")
'thwas was a string example....wow!!! thwas was really a string'
# oops - how to fix?
str1.replace(" is", " was")
'this was a string example....wow!!! this was really a string'
str1.replace(" is ", " was ")  # safer
```

String Methods

```
>>> 'species'.startswith('a')
False
>>> 'species'.startswith('spe')
True
>>> 'species'.startswith('Spe')
False
>>> str.upper('species').startswith('Spe')
False
>>> str.upper('species').startswith('SPE')
True
```

String Methods

```
# Formatting
center(...)
| S.center(width[, fillchar]) -> str

Return S centered in a string of length width. Padding is
done using the specified fill character (default is a space)

str.center('Table of Contents', 60)
'                    Table of Contents                     '

'Table of Contents'.center(60,'.') # pad with dots (periods)
'.....................Table of Contents.....................'
```

String Methods

```
# Stripping removes spaces AND newlines (\n)
compound = ' \n Methyl \n butanol \n'
print(compound)

 Methyl
 butanol

compound.strip()
'Methyl \n butanol'

print(compound.strip())
Methyl
 butanol
```

String.format Method

```
help(str.format)
Help on method_descriptor:
format(...)
    S.format(*args, **kwargs) -> str

    Return a formatted version of S, using substitutions from args and kwargs.
    The substitutions are identified by braces ('{' and '}').

# str.format indexes are inside curly braces which are inside quotation marks
print("{0}'s sister is named {1}".format('Tom','Sally') )
Tom's sister is named Sally

'"{0}" is derived from "{1}"'.format('none', 'no one')
'"none" is derived from "no one"'
```

String.format Method

```
my_pi = 3.14159

# Using format FUNCTION .2f = 2 decimal places
print("Pi rounded to 2 decimal places is", format(my_pi,'.2f'))
```

```
Pi rounded to 2 decimal places is 3.14
```

```
# Using str.format() uses index numbers starting with 0 inside curly braces

print('Pi rounded to {0} decimal places is {1}.'.format(2, my_pi))
```

```
Pi rounded to 2 decimal places is 3.14159.
```

String.format with flexible number format

```
# we can format numbers using the format method and function


# more flexible using a decimal places integer; notice no print() at front
decplace = 2

('Pi rounded to {0} decimal places is ' + '{1:.' + str(decplace) + 'f}.').format(decplace, my_pi)
'Pi rounded to 2 decimal places is 3.14.'

decplace = 3
('Pi rounded to {0} decimal places is ' + '{1:.' + str(decplace) + 'f}.').format(decplace, my_pi)
'Pi rounded to 3 decimal places is 3.142.'

decplace = 4
('Pi rounded to {0} decimal places is ' + '{1:.' + str(decplace) + 'f}.').format(decplace, my_pi)
'Pi rounded to 4 decimal places is 3.1416.'
```

String.format method and format Function

```
innum = 10
outnum = 16.1
inunit = 'miles'
outunit='km'

(format(outnum,'.1f') + ' {0} is equal to ' + format(innum,'.1f') + ' {1}').format(outunit, inunit)
'16.1 km is equal to 10.0 miles'
# easier to insert spaces above

format(outnum,'.1f') + outunit + ' is equal to ' + format(innum,'.1f') + inunit
'16.1km is equal to 10.0miles'
```

Homework 6B (second part)
Easy points!

Work with the module you chose and do 3 things

- run some demos
- try something new

(web scraper: go to a different web page)

Basically, make sure it works

Due Monday before class

Homework 7 due before class Monday

Exercises 7.6 on page 125 ff:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11

and...

Homework 7 due before class Monday

B. In the US, we calculate fuel efficiency by calculating miles per gallon (MPG). In Europe (and Canada), they calculate fuel efficiency by calculating liters per 100 km driven. Each makes sense, but they are not easy to convert in your head.

Write a function that will convert MPG to liters per 100 km, AND that will convert liters per 100 km to MPG, depending on an argument. Be sure to include units in your answer (mpg or l/100 km).

Example: ConvertMPGLp100(30, 'M2L')

Follow the function design recipe but include at least 6 example calls of correct output. At least one example needs to return what 41 MPG is in the European system.

Test the function using doctest.

Use 1 gallon = 3.79 liters, 1 mile = 1.61 km.

Provide an answer with 1 decimal place.