Journal for Lab1

Hongyan Yi (yih@oregonstate.edu)

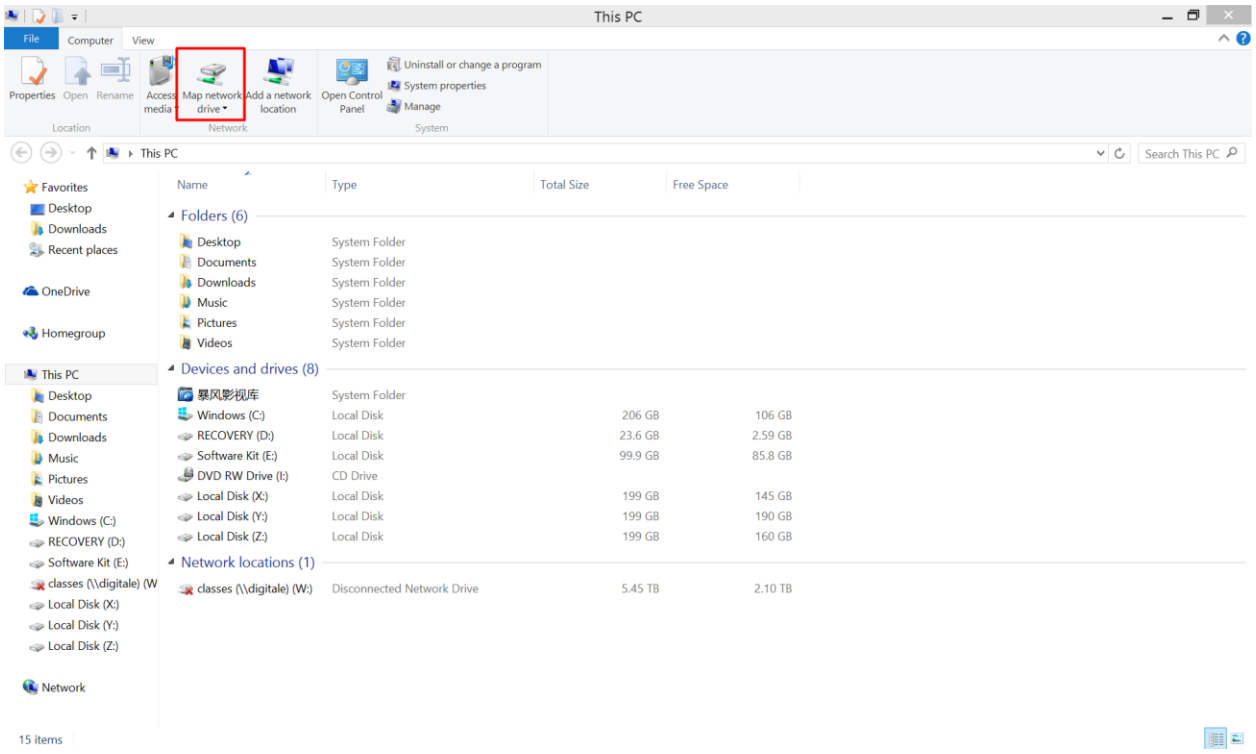
Main Goal

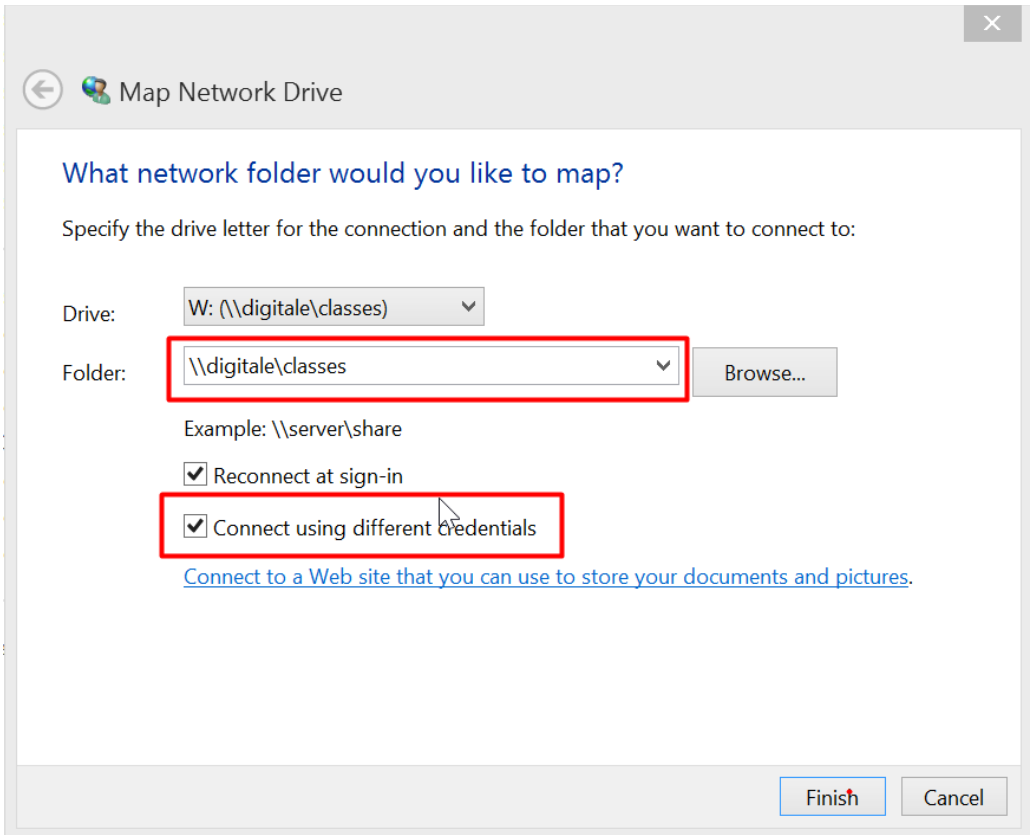
Notes for what I learned from three parts
1) Lab1 class
2) Codecademy thru Date and Time
3) Lab1 assignments

Lab1 Class

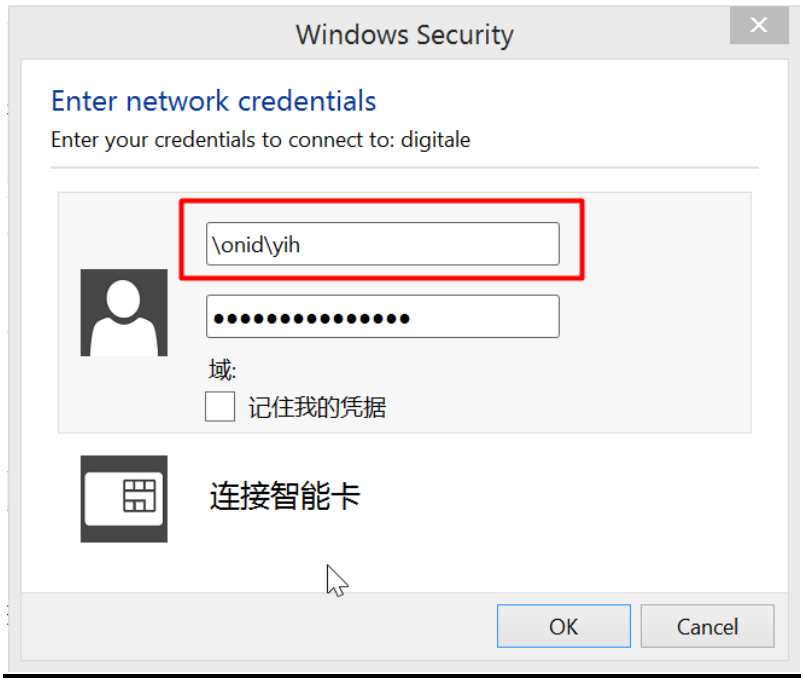
1. Setting environment

Connect to digitale with laptop





Folder: \\digitale\classes
username: onid\yih
password: password



2. Deliverables

Checklist of deliverables
At the end of lab, use Zip or 7Zip to bundle these files and place the zipped file in the Geo578/Drop/Lab1 folder:
Lab 1 Journal: <yourlastname>_<yourfirstname>_lab1_journal.docx or .pdf
Part 1: <yourlastname>_<yourfirstname>_lab1_part1.py
Part 2: <yourlastname>_<yourfirstname>_lab1_part2.py
Part 3: <yourlastname>_<yourfirstname>_lab1_part3.py

3. Global commands

execfile("yi_ hongyan_lab1_part1.py")

4. File Path

R:\Geo578\Students\yih\Lab1

5. Python Path

Lab: C:\Python27\ArcGIS10.2\python.exe
Laptop: C:\Python27\ArcGIS10.3\python.exe

6. Open .py with IDLE

IDLE: “Integrated Development Environment”, that gets bundled with the standard Python package. It is like a text editor, but adds more functionality to help you interpret your code.

Codecademy

1. multi-line comments

you can include the whole block in a set of triple quotation marks:

Multi-line Comments

Some comments need to span several lines, use this if you have more than 4 single line comments in a row.

Example

```
'''
this is
a multi-line
comment, i am handy for commenting out whole
chunks of code very fast
'''
```

2. raw input

```
1  name = raw_input("What is your name?")
2  quest = raw_input("What is your quest?")
3  color = raw_input("What is your favorite color?")
4
5  print "Ah, so your name is %s, your quest is %s, " \
6  "and your favorite color is %s." % (name, quest, color)
```

```
What is your name? angie
What is your quest? car
What is your favorite color? red
Ah, so your name is angie, your quest is
car, and your favorite color is red.
None
```

3. function space

You should indent your code with four spaces.

```
1  def spam():
2      eggs = 12
3      return eggs
4
5  print spam()
```

4. bool value

first character should be upper case

```
1  spam = True
2  eggs = False
```

5. percentage

The tax on your receipt is 6.75%. You'll have to divide 6.75 by 100 in order to get the decimal form of the percentage.

```
2  tax = 6.75 / 100
```

Lab Assignments

Part1

Subsection 1: Do some string examples.

task 1.1 copy of strings
Note: string could not multiply float.

```
>>> print "123456" * 5
123456123456123456123456123456
>>> print "123456" * 0.5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'float'
```

task 1.2: use backslash for special characters: apostrophe and backslash

ESCAPE	WHAT IT DOES.
\\	Backslash (\)
\'	Single-quote (')
\"	Double-quote (")
\a	ASCII bell (BEL)
\b	ASCII backspace (BS)
\f	ASCII formfeed (FF)
\n	ASCII linefeed (LF)
\N{name}	Character named name in the Unicode database (Unicode only)
\r	Carriage Return (CR)
\t	Horizontal Tab (TAB)
\uxxxx	Character with 16-bit hex value xxxx (Unicode only)
\Uxxxxxxxx	Character with 32-bit hex value xxxxxxxx (Unicode only)
\v	ASCII vertical tab (VT)
\ooo	Character with octal value ooo
\xhh	Character with hex value hh

task 1.3: string concatenation
two strings could directly catenate with +

Subsection 2: Accessing strings by index

task 2.1 Accessing strings by index

```
>>> print "0123456789"[1:3]
12
>>> print "0123456789"[1:]
123456789
>>> print "0123456789"[:3]
012
>>> print "0123456789"[-3:]
789
>>> print "0123456789"[:-3]
0123456
>>> print "0123456789"[-3:-1]
78
>>> print "0123456789"[:-1]
012345678
>>> print "0123456789"[-1:]
9
```

task 2.2 Dot notation

Methods that use dot notation only work with strings. For example: **string.upper() string.lower()** On the other hand, **len() and str()** can work on other data types.

```
lion = "roar"
len(lion)
lion.upper()
```

Subsection 3: Accessing strings by index: More advanced use!

As Python strings are **immutable**, and so, in order to modify a string you have to make use of the pieces you already have.

```
extension = "arecool"
longerfullfilename = "R:\\Geo578\\Students\\yih\\Lab1\\longer\\test.py"
index = longerfullfilename.find(".py")
strAfterInsert = longerfullfilename[:index] + extension + longerfullfilename[index:]
```

When inserting some string to another string, you could not directly use insert() or .insert(), but find the index and catenate them.

Subsection 4: Do some math examples:

Reference to: http://www.webreference.com/programming/python/index.html

Operation	Description
x + y	Addition
x - y	Subtraction
x * y	Multiplication
x / y	Division
x // y	Truncating division
x ** y	Power (x^y)
x % y	Modulo (x mod y)
-x	Unary minus
+x	Unary plus

Operation	Description
x << y	Left shift
x >> y	Right shift
x & y	Bitwise AND
x | y	Bitwise OR
x ^ y	Bitwise XOR (exclusive OR)
~x	Bitwise negation

Operation	Description
x < y	Less than
x > y	Greater than
x == y	Equal to
x != y	Not equal to (same as <>)
x >= y	Greater than or equal to
x <= y	Less than or equal to

Function	Description
abs(x)	Absolute value
divmod(x, y)	Returns (x // y, x % y)
pow(x, y [, modulo])	Returns (x ** y) % modulo
round(x, [n])	Rounds to the nearest multiple of 10^{-n} (floating-point numbers only)

Subsection 5: Decimals and more advanced calculations

task 5.1 Calculate the area of a circle

```
from __future__ import division
```

Use Python 3

In Python 3, to get true division, you simply do `a / b`.

Floor division, the classic division behavior for integers, is now `a // b`:

```
>>> 1/2
0.5
>>> 1//2
0
>>> 1//2.0
0.0
```

If Using Python 2

```
>>> from __future__ import division
>>> 1/2
0.5
>>> 1//2
0
>>> 1//2.0
0.0
```

This is really the best solution as it ensures you code is more forward compatible with Python 3.

Subsection 6,7: String formatting

Reference to: https://mkaz.github.io/2012/10/10/python-string-format/

would run: `print("{:.2f}".format(3.1415926));`

Number	Format	Output	Description
3.1415926	{:.2f}	3.14	2 decimal places
3.1415926	{:+.2f}	+3.14	2 decimal places with sign
-1	{:+.2f}	-1.00	2 decimal places with sign
2.71828	{:.0f}	3	No decimal places
5	{:0>2d}	05	Pad number with zeros (left padding, width 2)
5	{:x<4d}	5xxx	Pad number with x's (right padding, width 4)
10	{:x<4d}	10xx	Pad number with x's (right padding, width 4)
1000000	{:,}	1,000,000	Number format with comma separator
0.25	{:.2%}	25.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
13	{:10d}	13	Right aligned (default, width 10)
13	{:<10d}	13	Left aligned (width 10)
13	{:^10d}	13	Center aligned (width 10)

```
s1 = "so much depends upon {}".format("a red wheel barrow")
s2 = "glazed with {} water beside the {} chickens".format("rain", "white")
```

```
s1 = " {0} is better than {1} ".format("emacs", "vim")
s2 = " {1} is better than {0} ".format("emacs", "vim")
```

```
madlib = " I {verb} the {object} off the {place} ".format(verb="took", object="cheese", place="table")
~~ I took the cheese off the table
```

```
str = "Oh {0}, {0}! wherefore art thou {0}?".format("Romeo")
~~ Oh Romeo, Romeo! wherefore art thou Romeo?
```

```
print("{0:d} - {0:x} - {0:o} - {0:b} ".format(21))
~~ 21 - 15 - 25 - 10101
```

```
print(" The {} set is often represented as {{0}}".format("empty"))
~~ The empty set is often represented as {0}
```

Subsection 8: Converting formats

Use str() and .replace()

Subsection 9: Working with Date and Time

task9.1 Load the datetime module using import.

```
from datetime import datetime
starttime = datetime.now()      # now is not a property but a function
```

Subsection 10: New functions

https://docs.python.org/2/library/functions.html#type

		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

class `list`([*iterable*])

Return a list whose items are the same and in the same order as *iterable*'s items. *iterable* may be either a sequence, a container that supports iteration, or an iterator object. If *iterable* is already a list, a copy is made and returned, similar to `iterable[:]`. For instance, `list('abc')` returns `['a', 'b', 'c']` and `list( (1, 2, 3) )` returns `[1, 2, 3]`. If no argument is given, returns a new empty list, `[]`.

`list` is a mutable sequence type, as documented in Sequence Types — str, unicode, list, tuple, bytearray, buffer, xrange. For other containers see the built in `dict`, `set`, and `tuple` classes, and the `collections` module.

`enumerate`(*sequence*, *start=0*)

Return an enumerate object. *sequence* must be a sequence, an iterator, or some other object which supports iteration. The `next()` method of the iterator returned by `enumerate()` returns a tuple containing a count (from *start* which defaults to 0) and the values obtained from iterating over *sequence*:

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

Equivalent to:

```
def enumerate(sequence, start=0):
    n = start
    for elem in sequence:
        yield n, elem
        n += 1
```

New in version 2.3.

Changed in version 2.6: The *start* parameter was added.

Part2

Note: no necessary

Part3

1. convert int to date

timedelta is a good way to convert any int time(days, seconds,...) to formal date format.

```
# task3: caculate and show months delta of two date
# algorithm: find the days of d1.month,add the days to d1 until d1<=d2, the times of loop is the month delta
def monthdelta(d1, d2):     # by defalut d1 < d2
    delta = 0
    while True:             # use loop: while True,if-else-break
        mdays = monthrange(d1.year, d1.month)[1]    # calendar.monthrange(year, month)Returns weekday of first day
        d1 += timedelta(days = mdays)               # convert int to date: datetime.timedelta, mdays datatype is i
        if d1 <= d2:
            delta += 1
        else:
            break
    return delta

month_delta = monthdelta(birth_date,end_of_term)
print "After this term, Alex will be %d months old" %month_delta
```

class `datetime.timedelta`([*days*[, *seconds*[, *microseconds*[, *milliseconds*[, *minutes*[, *hours*[, *weeks*]]]]]]])

All arguments are optional and default to `0`. Arguments may be ints, longs, or floats, and may be positive or negative.

Only *days*, *seconds* and *microseconds* are stored internally. Arguments are converted to those units:

- A millisecond is converted to 1000 microseconds.
- A minute is converted to 60 seconds.
- An hour is converted to 3600 seconds.
- A week is converted to 7 days.

```
>>> print timedelta(days = 600)
600 days, 0:00:00
>>> print timedelta(days = 0.01)
0:14:24
>>> print timedelta(seconds = 0.01)
0:00:00.010000
>>> print timedelta(seconds = 9)
0:00:09
>>> print timedelta(weeks = 0.1)
16:48:00
>>> print timedelta(weeks = 5.1)
35 days, 16:48:00
>>> print timedelta(weeks = 5.6)
39 days, 4:48:00
>>> print timedelta(weeks = 5.56)
38 days, 22:04:48
>>> print timedelta(weeks = 5)
35 days, 0:00:00
>>> print timedelta(year = 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'year' is an invalid keyword argument for this function
>>> print timedelta(hours= 5)
5:00:00
>>> print timedelta(hours= 5.3)
5:18:00
```

```
>>> print monthrange(2016,1)
(4, 31)
>>> print monthrange(2016,2)
(0, 29)
>>> print monthrange(2016,3)
(1, 31)
>>> print monthrange(2016,4)
(4, 30)
```

The second value returns are right, but the first sometimes is wrong for example

```
>>> print monthrange(2016,2)
(0, 29)
>>> print monthrange(2015,2)
(6, 28)
>>> print monthrange(2015,3)
(6, 31)
>>> print monthrange(2015,4)
(2, 30)
```

2. convert date to int

any date variable could use its attribute to get exact int value of days.

```
today_date = date.today()          # Return the current local datetime, with tzinfo None.
end_of_term = date(2016,3,18)
interval = end_of_term - today_date
interval_days = interval.days       # convert date to int
print "This term has %d days left" %(interval_days)
```

3. datetime.today() differ date.today()

classmethod datetime.today()

Return the current local datetime, with tzinfo None. This is equivalent to datetime.fromtimestamp(time.time()). See also now(), fromtimestamp().

classmethod date.today()

Return the current local date. This is equivalent to date.fromtimestamp(time.time()).

```
>>> from datetime import datetime
>>> print datetime.today() - datetime.date(2016,1,1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: descriptor 'date' requires a 'datetime.datetime' object but received a 'int'
>>>
```

```
>>> from datetime import date
>>> print date.today() - date(2016,1,1)
9 days, 0:00:00
```

4. fomat of date(year,month,day)

from datetime import date
date(2015,09,18) #wrong with 09
date(2015,9,18)
date(1985,12,05) #right with 05