Get Started - Setting up your Python Development Environment

Lesson 1 - PythonAnywhere & Installing Python

- PythonAnywhere (Cloud/browser - no installation)
- Setting up Python environment in Microsoft Windows
- Setting up Python environment on a Macintosh
- Can also set up Win-10 using Windows Subsetsystem for Linux (WSL) if you prefer a Linux-like experience on computer
- Others: (e.g.) Trinket, Cloud 9, CodeAnywhere

① PythonAnywhere 1.1 Files Tab 1.2 Bash Console e.g.: 14:12 ~ $ cd

1.2.1 Commands: · cd - Change directory into target folder (to ensure right folder hierarchy)
· pwd - Print Work Directory → tell where you are at in the folder hierarchy
· ls -l - list files & subfolders in the current folder. The "-l" option shows details like permissions, modification date and file size.

② Python for Windows 2.1 Installation → www.python.org/download/ ← remember check "Add Python 3.X to PATH"

2.2 Install the Atom Text Editor → atom.io ← Can choose other text editor

2.3 Command Prompt (命令提示符) → Search in Win-10
- Input "python" to enter edit mode
- Use Ctrl + Z Buttons to exit the mode

Two Windows
2.4 Use Atom Editor to creat a new .py file & save → Editor highlights syntax
2.5 Command Prompt →
- cd desktop → Go to "desktop" sub-directory.
- dir → show folders under current directory
- cd XXX → Go to "XXX" sub-directory
- cd → show current directory
- python first.py / first.py → run first.py (in correct directory)
- "Tab" Button → auto complete folder name by syste
- "↑" Arrow Button → scroll back through previous commands

Home directory → C:\Users\MINRUI>

Top left corner of Window → Preference → Increase Buffer Size (999)

Lesson 2 - Why program

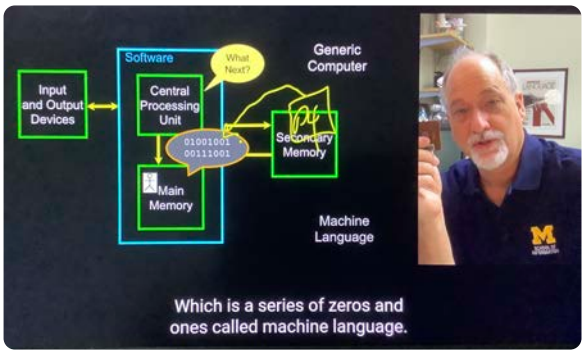
"Not machines try to speak our languages, but we learn to speak machines' languages"

- To get some task done - we are end users & programmers
- To produce something for others to use - a programming job

What is code? → · a sequence of stored instructions · a piece of creative art

Computer is extremely literal & we have to be very precise ← what we mean = what we say/code

→ Hardware Architecture [Raspberry Pi (樹莓派, 迷你計算機系統)]
- Central processing unit (CPU)
- Main Memory (RAM)
- Mother Board
- Secondary Memory (Hard drive)
- Input Devices
- Output Devices



Syntax Errors

Elements of Python: { Vocabulary/Words; Sentence structure; Story structure }

Reserved Words

Reserved Words

- You cannot use reserved words as variable names / identifiers

False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
and	del	global	not	with
as	elif	try	or	yield
assert	else	import	pass	
break	except	in	raise	

Variable Operator Constant Reserved word

Interactive → type directly to Python one line at a time and it responds

Python Scripts → XXX.py → type a sequence of statements/lines into a file using a text editor and execute

Program Steps/Flow: Sequential; Repeated; Conditional

```
name = input('Enter file:')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Sequential
Repeated
Conditional

```
name = input('Enter file:')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

A short Python "Story" about how to count words in a file

A word used to read data from a user

A sentence about updating one of the many counts

A paragraph about how to find the largest item in a list

Lesson 3 - Variables, expressions, and statements

Mnemonic Variable names → can name variables as we like (But not reserved words)

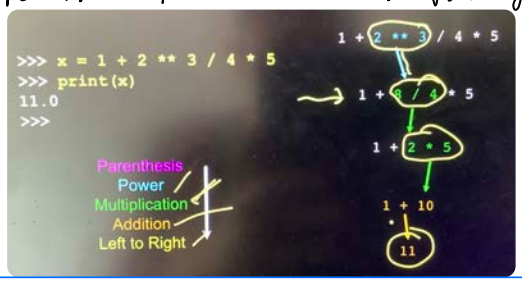
Expressions → ① Numeric Expressions → +, -, *, /, **, %

e.g. jj=23 kk=jj%5 (result is "3") Power Remainder (integer division)

※ Order of Evaluation → Parenthesis, Power, Multiplication, Addition, Left to right (括弧 冪)

e.g.

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```



② Type → Python knows difference between an integer number and a string

e.g.: "+" < "Addition" if numbers / "Concatenate" if strings

※ No number + string is recognised (error)

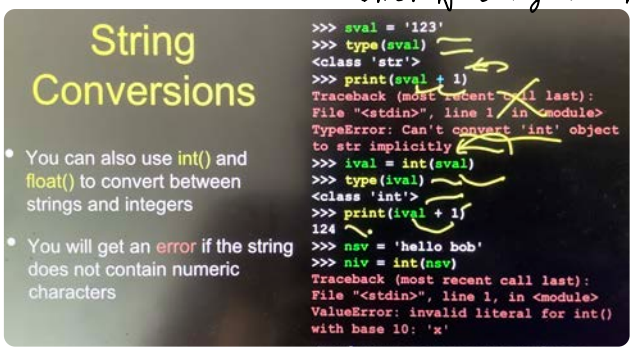
e.g.: eee = eee + 1 is an error statement in Python

※ >>> type( ) → what type something is | · Types of numbers → Integers -14
| Floating Point Numbers -14.0

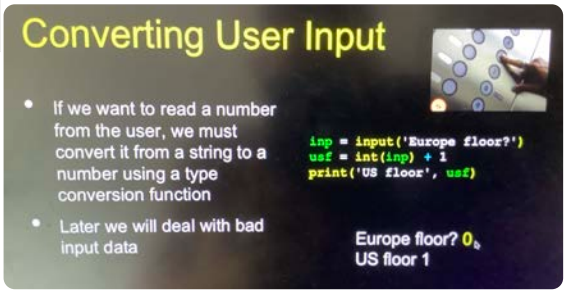
· Type conversion → Integer + Float → implicitly convert to a float

· >>> print (XXX/XXX) → e.g. >>> print (10/2) | >>> print (99.0//00.0)
5.0 | 0.99

③ String Conversions → · can use int( ) & float( ) to convert between strings & Integers
· error if string does not contain numeric characters



④ >>> input( )
→ returns a string
e.g. nam = input ('Who are you')
print = ('Welcome', nam)
result: Who are you? MIN RUI
Welcome, MIN RUI

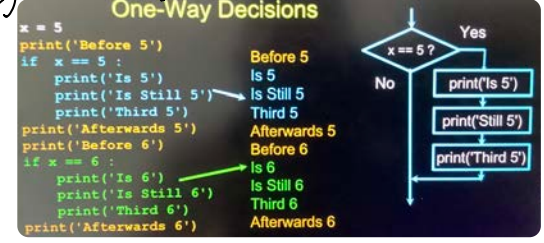


Comments in Python → ※ Anything after # is ignored by Python
- Describe code
- Document info
- Turn off a line of code (perhaps temporarily)

```
# Get the name of the file and open it
name = input('Enter file:')
handle = open(name, 'r')

# Count word frequency
counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

# Find the most common word
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

# All done
print(bigword, bigcount)
```

Lesson 4 - Conditional Execution "If statement"

Comparison Operators → · < · <= · == · >= · > · !=
(Note: "=" is for assigning purpose) (equal to) (Not equal)



Indentation (缩进) · Blanks are important · Avoid using real "Tab" Button

Indentation

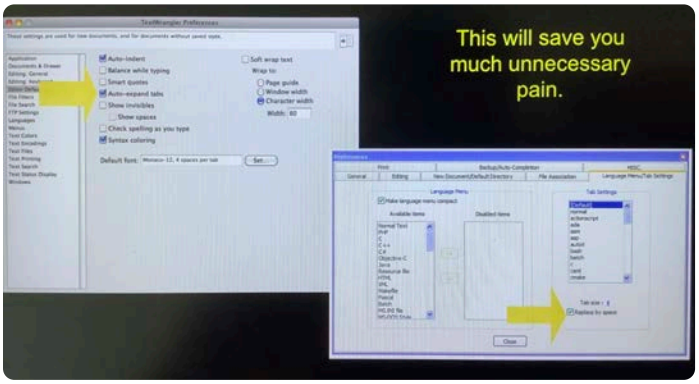
- Increase indent indent after an if statement or for statement (after :)
- Maintain indent to indicate the scope of the block (which lines are affected by the if/for)
- Reduce indent back to the level of the if statement or for statement to indicate the end of the block
- Blank lines are ignored - they do not affect indentation
- Comments on a line by themselves are ignored with regard to indentation

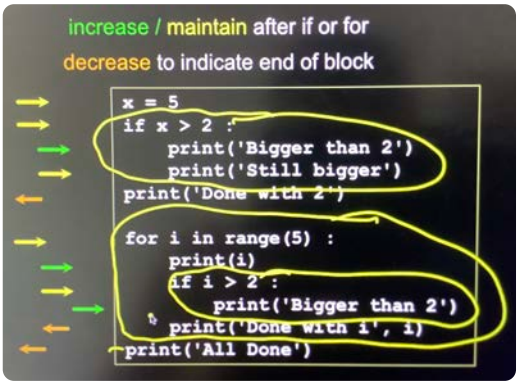
after an if, you must indent,

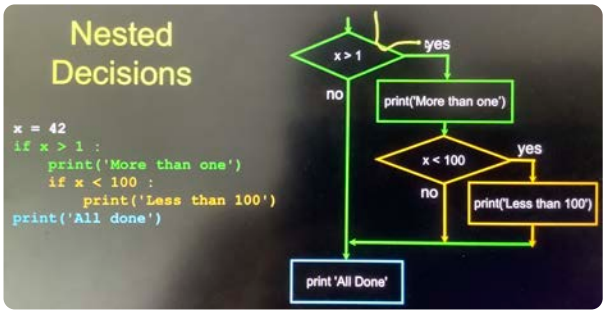
Warning: Turn Off Tabs!

- Atom automatically uses spaces for files with ".py" extension (nice!)
- Most text editors can turn tabs into spaces - make sure to enable this feature
  - NotePad++: Settings -> Preferences -> Language Menu/Tab Settings
  - TextWrangler: TextWrangler -> Preferences -> Editor Defaults
- Python cares a *lot* about how far a line is indented. If you mix tabs and spaces, you may get "indentation errors" even if everything looks fine

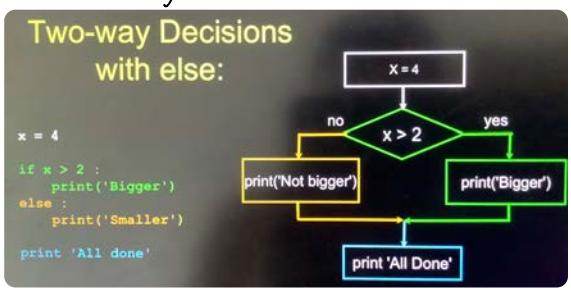
so we tend to avoid using actual tabs in files.



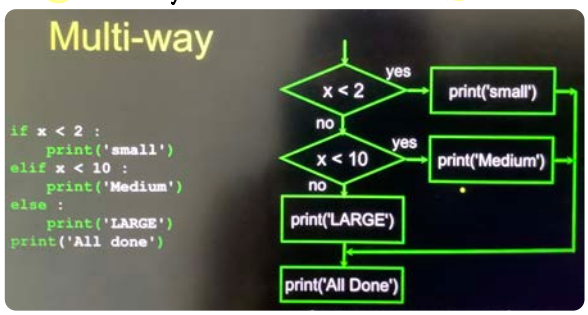


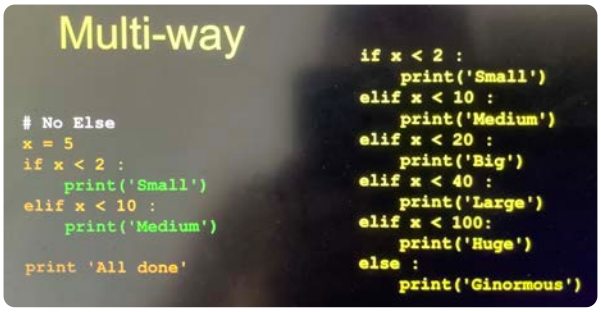


Two-way Decision with else:

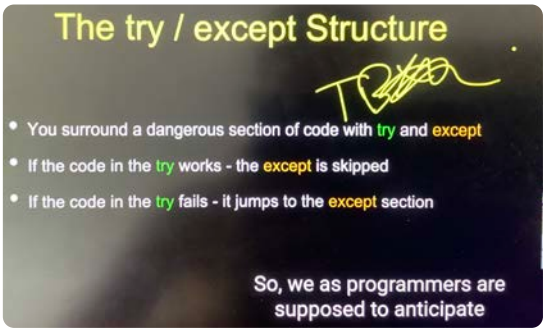


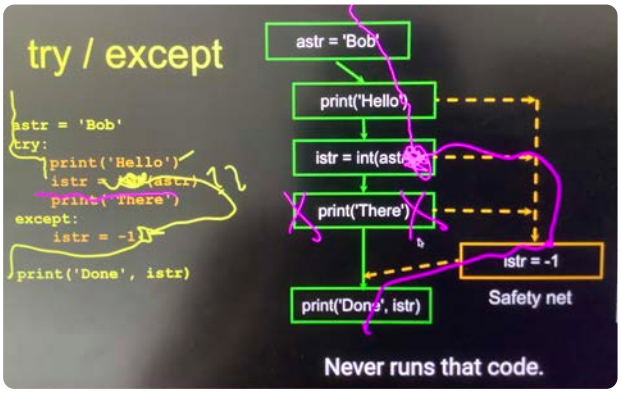
Multi-way Decisions — "elif"





Try / except structure (avoid trace back)





>>> quit() ← can be used under except:

So do not put unnecessary lines under "try:"

Lesson 5 – Functions "Store & reuse"

2 types { Built-in functions ①
functions that we define ② def XXX(): → define a function
then we call/invoke the function

① e.g.: big = max('Hello World') → 'w' is assigned
(assignment, argument, built-in function)
※ lowercase letters are bigger than uppercase ones
tiny = min('Hello World') → ' ' is assigned (blank)

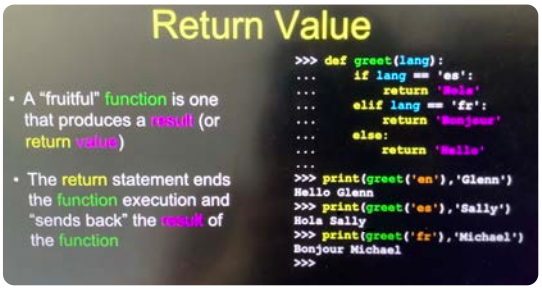
② Defined functions 2.1 def XXX(): → define a function (no output)
2.2 XXX() → call/invoke the function
} 2 steps!

e.g.:
```
>>> def greet(lang):
        if lang == 'es':
            print('Hola')
        elif lang == 'fr':
            print('Bonjour')
        else:
            print('Hello')
```

---

return XXX

※



Multiple Parameters / Arguments

e.g.:

```
def add_two(a, b):
    added = a + b
    return added
```

Void (non-fruitful) Functions

Lesson 6 - Loops and Iteration

① "while" loops (An infinite loop)

e.g.:

```
while n > 0 :
    print(n)
    n = n - 1
```

1-1 Breaking out of a loop — **break**

ends the current loop and jumps to the statement immediately following the loop

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

1-2 Finishing an iteration with continue — **continue** → ends the current iteration & jump to the top of the loop

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

② "for" loops (Definite loop)

e.g.:

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print ('Blast off!')
```

iteration variable

five-element sequence

e.g.:

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:', friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

③ Loop Idioms : what we do in loops

e.g.:

Finding the largest value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74
```

- Set some variables to initial values ← Before the loop starts
- for thing in data:
  - Look for something or do something to each entry separately, updating a variable ← During the loop
- Look at the variables ← What you get / how to use that after the loop

④ More patterns of different things we can do through loops

Counting in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

Summing in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

Finding the Average in a Loop

```
count = 0
sum = 0
print('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('After', count, sum, sum / count)
```

```
$ python averageloop.py
Before 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
After 6 154 25.666
```

Filtering in a Loop

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print('Large number',value)
print('After')
```

```
$ python search1.py
Before
Large number 41
Large number 74
After
```

True ; False → (type: Boolean)

None → a constant (type: none) to indicate emptiness

Finding the smallest value

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

```
$ python smallest.py
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

The "is" and "is not" Operators

```
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value

print('After', smallest)
```

0 == 0.0

0 is 0.0

- Python has an is operator that can be used in logical expressions
- Implies "is the same as"
- Similar to, but stronger than ==
- is not also is a logical operator

these two are not the same type wise.

{ 0 == 0.0 correct (value wise)
{ 0 is 0.0 wrong (value & type wise)

"is" ; "is not"

※ Don't overuse "is"/"is not" to avoid confusing
→ tend to only use on Booleans & None types

Lesson 7 – Strings

Reading and Converting

① Looking inside strings (from index 0)

b	a	n	a	n	a
0	1	2	3	4	5

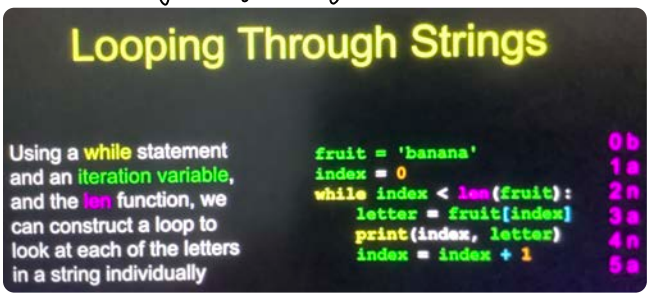
② A character Too Far → python error (traceback)

e.g.: >>> zot = 'abc'
>>> print (zot[5])

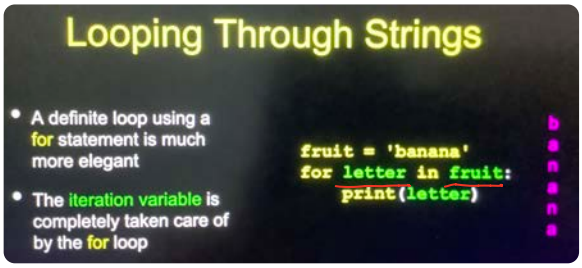
④ Strings have length → e.g.: >>> fruit = 'banana'
>>> print (len (fruit))
6

len() → length

⑤ Looping through strings



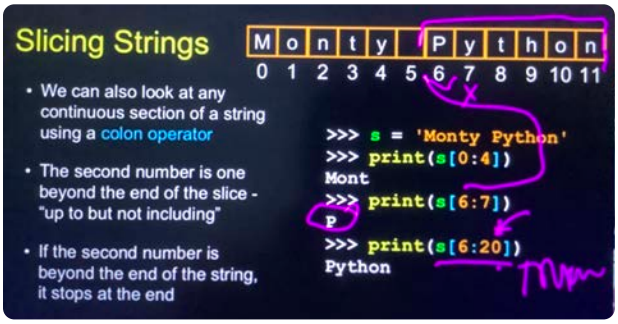
Indeterminant Loop
(note: index starts from 0)



Determinant Loop (more elegant)

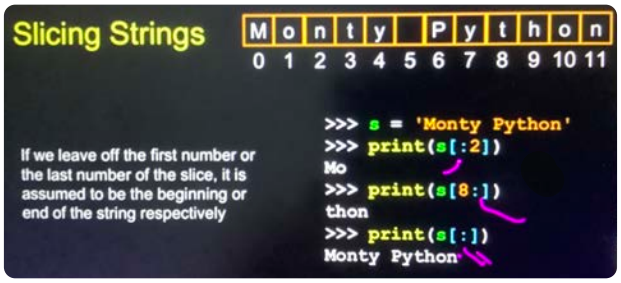
⑥ Slicing strings

M o n t y P y t h o n
0 1 2 3 4 5 6 7 8 9 10 11



S[a:b] include a^{th} word but Not b^{th} word

if b > len(s) NO traceback happens



⑦ String Concatenation (串联)

※ print(x, y) will automatically create a space between x and y
BUT + operator will Not

String Concatenation

When the + operator is applied to strings, it means "concatenation"

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

print(x, y)

⑧ Using in as a logical operator

Using in as a logical Operator

- The in keyword can also be used to check to see if one string is "in" another string
- The in expression is a logical expression that returns True or False and can be used in an if statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

⑨ String Comparison (hard to predict output)

generally, Lowercase > uppercase (but may change through setting)

⑩ String Library (Can use >>> print(dir(XXX)) to check list)

Some Commands

type( ) → show type of content

dir( ) → show all the different methods/capabilities/things we can do to strings

s.XXX( ) ← (object-oriented function)

String Library

```
str.capitalize()
str.center(width[, fillchar])
str.endswith(suffix[, start[, end]])
str.find(sub[, start[, end]])
str.lstrip([chars])
str.replace(old, new[, count])
str.lower()
str.rstrip([chars])
str.strip([chars])
str.upper()
```

Searching a String

- We use the find() function to search for a substring within another string
- find() finds the first occurrence of the substring
- If the substring is not found, find() returns -1
- Remember that string position starts at zero

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

Making everything UPPER CASE

- You can make a copy of a string in lower case or upper case
- Often when we are searching for a string using find() we first convert the string to lower case so we can search a string regardless of case

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print(nnn)
HELLO BOB
>>> www = greet.lower()
>>> print(www)
hello bob
>>>
```

- Python has a number of string functions which are in the string library
- These functions are already built into every string - we invoke them by appending the function to the string variable
- These functions do not modify the original string, instead they return a new string that has been altered

Search and Replace

- The replace() function is like a "search and replace" operation in a word processor
- It replaces all occurrences of the search string with the replacement string

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob','Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o','X')
>>> print(nstr)
HellX BXb
>>>
```

replace all, not just first

Stripping Whitespace

TAB NEWLINE

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- lstrip() and rstrip() remove whitespace at the left or right
- strip() removes both beginning and ending whitespace

```
>>> greet = '   Hello Bob   '
>>> greet.lstrip()
'Hello Bob   '
>>> greet.rstrip()
'   Hello Bob'
>>> greet.strip()
'Hello Bob'
>>>
```

Prefixes

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

Parsing and Extracting

21 31

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```

find ' ' starting from the position atpos

Two Kinds of Strings

```
Python 3.5.1
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

```
Python 2.7.10
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

In Python 3, all strings are Unicode

It is good that Python 3 can recognise all non-Latin characters as strings

---

Lesson 8 - Files

Reading Files - mostly .txt files

① Opening a File

open( ) → returns a "file handle" (similar to "File → Open")

open(filename, mode)

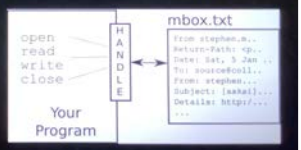
- filename: name of the file
- mode: "r" → read the file; "w" → write the file

When Files are Missing

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or
directory: 'stuff.txt'
```

What is a Handle?

```
>>> fhand = open('mbox.txt')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='UTF-8'>
```



It doesn't print the lines that are in the file.

② "\n" (backslash n) → "newline" to indicate when a line ends
it occupies ONE space (but we cannot see it, a kind of white space)

The newline Character

- We use a special character called the "newline" to indicate when a line ends
- We represent it as \n in strings
- Newline is still one character - not two

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print(stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

File Processing

A text file has newlines at the end of each line

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008\n
Return-Path: <postmaster@collab.sakaiproject.org>\n
Date: Sat, 5 Jan 2008 09:12:18 -0500\n
To: source@collab.sakaiproject.org\n
From: stephen.marquard@uct.ac.za\n
Subject: [sakai] svn commit: r39772 - content/branches/\n
\n
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\n
```

↑every line is ended by a newline "\n"

③ Read - File Handle as a Sequence ("for" loops)

File Handle as a Sequence

- A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence
- We can use the for statement to iterate through a sequence
- Remember - a sequence is an ordered set

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

← 实质上会列印出所有的 lines

④ Read - Counting Lines in a File ("for" loops)

Counting Lines in a File

- Open a file read-only
- Use a for loop to read each line
- Count the lines and print out the number of lines

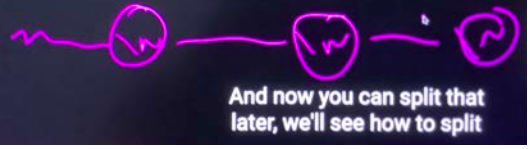
```
fhand = open('mbox.txt')
count = 0
for line in fhand:
    count = count + 1
print('Line Count:', count)

$ python open.py
Line Count: 132045
```

⑤ Read the *Whole* File → S.read()

Reading the *Whole* File

We can read the whole file (newlines and all) into a single string

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```



And now you can split that later, we'll see how to split

⑥ Searching through a File ("if" statement(s) in "for" loop)

Searching Through a File

We can put an if statement in our for loop to only print lines that meet some criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:') :
        print(line)
```

the problem is print() adds newline to each line found

Searching Through a File (fixed)

- We can strip the whitespace from the right-hand side of the string using rstrip() from the string library
- The newline is considered "white space" and is stripped

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....

.rstrip() gets rid of each line's "\n"

Skipping with continue

We can conveniently skip a line by using the continue statement

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:') :
        continue
    print(line)
```

↑ Another way to do the same thing

⑦ Using "in" to select lines

Using in to select lines

We can look for a string anywhere in a line as our selection criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not '@uct.ac.za' in line :
        continue
    print(line)
```

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f...
```

⑧ Prompt for File name & try bad File Names

Prompt for File Name

← use input()

```
fname = input('Enter the file name:  ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

Enter the file name: mbox-short.txt
There were 27 subject lines in mbox-short.txt

Bad File Names

```
fname = input('Enter the file name:  ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    quit()

count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

Good name → Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

Bad name → Enter the file name: na na boo boo
File cannot be opened: na na boo boo