301AA - Advanced Programming

Lecturer: **Andrea Corradini**
andrea@di.unipi.it
http://pages.di.unipi.it/corradini/

Course pages:
http://pages.di.unipi.it/corradini/Didattica/AP-18/

***AP-2017-29**: Libraries and Scritping in Python*

We have seen:

- Basic and Sequence Datatypes
- Dictionaries
- Control Structures
- List Comprehension
- Function definition
- Positional and keyword arguments of functions
- Namespaces and Scopes
- Object Oriented programming in Python
- Inheritance
- Iterators and generators
- Functions as objects
- Higher-order functions
- Importing modules
- More on higher-order functions
- Decorators
- Garbage collection and GIL
- Other criticisms to Python

Summary

We will browse the Python on-line documentation to get a grasp on where to look for Python features and available extension modules when using it a scripting language

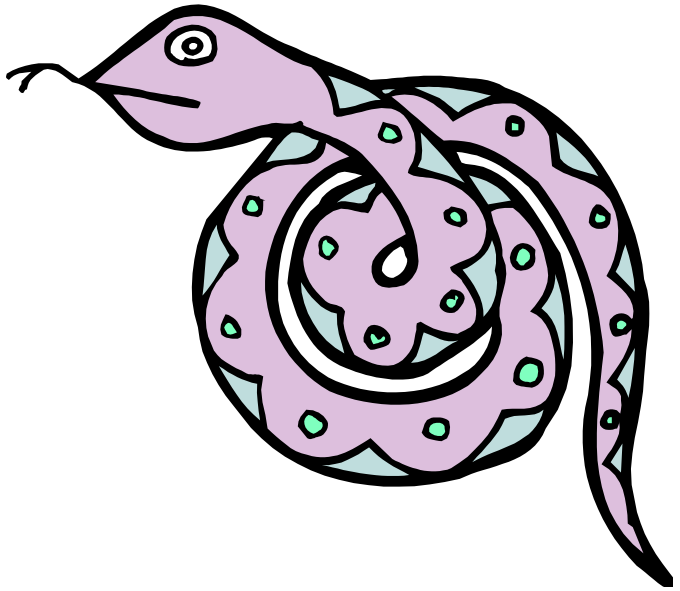
- **Builtin Functions**
  - https://docs.python.org/3.7/library/functions.html
- **Standard library**
  - https://docs.python.org/3.7/tutorial/stdlib.html

Next we will analyse some simple Python scripts providing some higher-level access to functionalities of the underlying operating systems, commenting mainly on the programming style and on portability issues.

Python Scripts for System Administrators

https://www.ibm.com/developerworks/aix/library/au-python/index.html

The next slides present tested Python 3 code essentially equivalent to the examples in the URL above



Example 1: Search for files and show permissions in a friendly format

```
import stat, sys, os, subprocess            # Python 3

# Getting search pattern from user and assigning it to a list

try:
    # run a 'find' command and assign results to a variable
    pattern = input("Enter the file pattern to search for:\n")
    commandString = "find " + pattern
    commandOutput = subprocess.getoutput(commandString)
    findResults = commandOutput.split("\n")

    # output find results, along with permissions
    print ("Files:")
    print (commandOutput)
    print ("================================")
    for file in findResults:
        mode = stat.S_IMODE(os.lstat(file)[stat.ST_MODE])
        print ("\nPermissions for file ", file, ":")
        for level in "USR", "GRP", "OTH":
            for perm in "R", "W", "X":
                if mode & getattr(stat,"S_I"+perm+level): # bitwise and
                    print (level, " has ", perm, " permission")
                else:
                    print (level, " does NOT have ", perm, " permission")
except:
    print ("There was a problem - check the message above")
```

Example 2: Perform operations on a tar archive that is based on menu selection

```
import tarfile, sys                 # Python 3

try:
    #open tarfile
    tar = tarfile.open(sys.argv[1], "r:tar")    # <class 'tarfile.TarFile'>

    #present menu and get selection
    selection = input("Enter\n\
    1 to extract a file\n\
    2 to display information on a file in the archive\n\
    3 to list all the files in the archive\n\n")

    #perform actions based on selection above
    if selection == "1":
        filename = input("enter the filename to extract:  ")
        tar.extract(filename)
    elif selection == "2":
        filename = input("enter the filename to inspect:  ")
        for tarinfo in tar:                 # <class 'tarfile.TarInfo'>
            if tarinfo.name == filename:
                print( "\n\
                Filename:\t\t", tarinfo.name, "\n\
                Size:\t\t", tarinfo.size, "bytes\n")
    elif selection == "3":
        print (tar.list(verbose=True))
except:
    print ("There was a problem running the program")
```

Example 3: Check for a running process and show information in a friendly format

```
import subprocess, os                    # Python 3

program = input("Enter the name of the program to check: ")

try:
    #perform a ps command and assign results to a list
    output = subprocess.getoutput("ps -f|grep " + program)
    processes = output.split("\n")
    for process in processes:
        proginfo = process.split()
        #display results
        print ("\n\                                    # Note: correctness depends
        Full path:\t\t", proginfo[7], "\n\             # on the structure of the
        Owner:\t\t\t", proginfo[0], "\n\               # output of ps
        Process ID:\t\t", proginfo[1], "\n\
        Parent process ID:\t", proginfo[2], "\n\
        Time started:\t\t", proginfo[4], "\n\
        **************************")
except:
    print ("There was a problem with the program.")
```

Example 4: Check userids and passwords for policy compliance

```
import pwd
#initialize lists
erroruser = []
errorpass = []
#get password database
passwd_db = pwd.getpwall()          # a list of <class 'pwd.struct_passwd'>
try:
    #check each user and password for validity
    for entry in passwd_db:         # <class 'pwd.struct_passwd'>
        username = entry[0]         # also  entry.pw_name
        password = entry [1]        # also  entry.pw_passwd
        if len(username) < 6:
            erroruser.append(username)
        if len(password) < 8:
            errorpass.append(username)
    #print results to screen
    print ("The following users have an invalid userid (< six characters):")
    for item in erroruser:
        print (item)
    print ("\nThe following users have invalid password(< eight characters):")
    for item in errorpass:
        print (item)
except:
    print ("There was a problem running the script.")
```

Example 5: Force "quit" of selected processes (from Ch 13 of Scott: Programming Language Pragmatics)

```
import sys, os, re, time
if len(sys.argv) != 2:
    sys.stderr.write('usage: ' + sys.argv[0] + ' pattern\n')
    sys.exit(1)
PS = os.popen("/bin/ps -w -w -x -o'pid,command'") # opens a pipe!
line = PS.readline()                  # discard header line
line = PS.readline().rstrip()         # prime pump
while line != "":
    proc = int(re.search('\S+', line).group()) # first occurrence of non-blanks
    if re.search(sys.argv[1], line) and proc != os.getpid():
        print (line + '? ',end='', flush=True)
        answer = sys.stdin.readline()
        while not re.search('^[yn]', answer, re.I):
            print ('? ',end='', flush=True)
            answer = sys.stdin.readline()
        if re.search('^y', answer, re.I):
            os.kill(proc, 9)
            time.sleep(1)
            try:                      # expect exception if process
                os.kill(proc, 0)      # no longer exists
                sys.stderr.write("unsuccessful; sorry\n"); sys.exit(1)
            except: pass          # do nothing
        sys.stdout.write('')      # inhibit prepended blank on next print
    line = PS.readline().rstrip()
```

Example: Create a PDF file containing the source code of a list of Java, Haskell and Python files (uses 'a2ps' and 'ps2pdf')

➔ **compare with the next script**

```
import sys, subprocess
# checks the number of arguments
if len(sys.argv) != 3:
    sys.stderr.write('usage: ' + sys.argv[0] + ' paths_file ps_file_name\n')
    sys.exit(1)
# reads the file, assuming that it contains relative paths of source files
paths_file = open(sys.argv[1])
#strips "\n" from each file name
stripped = (f.strip("\n") for f in  paths_file)
# filters out files of wrong type (with suffix not in {.java,.py,.hs})
checked = (file for file in stripped if file.endswith((".java",".hs",".py")))
# concatenates the file names
a2ps_files = " ".join(checked)
# using the 'a2ps' utility, generates a single PostScript file containing
# a pretty printed version of all the files passed as first argument.
command = "a2ps -A fill -o "+ sys.argv[2] + ".ps " +  a2ps_files
result = subprocess.getstatusoutput(command)
if result[0]!= 0 :
    print("There was an error... Result of a2ps: " + result[1])
else :
    print("Result of a2ps: " + result[1])
# converts a PostScript file to PDF using 'ps2pdf'
command = "ps2pdf "+ sys.argv[2] + ".ps "
result = subprocess.getstatusoutput(command)
print("Postscript file " + (sys.argv[2] + ".ps") + " converted to PDF.")
```

Example: The previous script, structured as a set of functions corresponding to basic operations. It can be invoked in the interpreter or as command line arg to **python**

```
import sys, subprocess

def check_args() -> None :
    # checks the number of arguments
    if len(sys.argv) != 3:
       sys.stderr.write('usage: ' + sys.argv[0] + ' paths_file ps_file_name\n')
       sys.exit(1)
def prepare_a2ps_args(paths_file:str) -> str :
    # builds a string listing the files to be printed
    # reads the file, assuming that it contains relative paths of source files
    pathsFile = open(paths_file)
    stripped = (f.strip("\n") for f in paths_file) # strips "\n" from file name
    # filters out files of wrong type (with suffix not in {.java,.py,.hs})
    checked = (fi for fi in stripped if fi.endswith((".java",".hs",".py")))
    # concatenates the file names
    return " ".join(checked)

def generate_postscript(file_names:[str], ps_file_name: str) -> str :
    # using the 'a2ps' utility, generates a single PostScript file containing
    # a pretty printed version of all the files passed as first argument.
    command = "a2ps -A fill -o "+ ps_file_name + ".ps " +  file_names
    result = subprocess.getstatusoutput(command)
    if result[0]!= 0 :
        return ("c'e' stato un errore... Result of a2ps: " + result[2])
    else :
        return ("Result of a2ps: " + result[1])
```

Example: The previous script, structured as a set of functions corresponding to basic operations. (cont.)

```
# (continue)

def generate_PDF(ps_file_name: str) -> str :
    # converts a PostScript file to PDF using 'ps2pdf'
    command = "ps2pdf "+ ps_file_name + ".ps "
    result = subprocess.getstatusoutput(command)
    print("Postscript file " + (sys.argv[2] + ".ps") + " converted to PDF.")

def main():
    # to be run in the interpreter: arguments are asked interactively
    file_list = input("Name of file with list of files to print? ")
    ps_name = input("Name of PS/PDF file? ")
    a2ps_args = prepare_A2psArgs(file_list)
    print(generate_postscript(a2ps_args, ps_name))
    print(generate_PDF(psName))

if __name__ == "__main__" :
    # executed when passed as argument to 'python3'
    check_args()
    a2psArgs = prepare_a2ps_args(sys.argv[1])
    print(generate_postscript(a2ps_args, sys.argv[2]))
    print(generate_PDF(sys.argv[2]))
```

Concluding remarks

When writing scripts, try hard to meet the following goals:

- **Portability**: make as few assumptions as possible on the underlying operating system, possibly none
- **Readability:** comment the code, and annotate function arguments and the function result with the expected types
- **Reusability**: the script should be made of a set of functions, each implementing a small and well identified task
- **Executability**: make the script executable in a variety of modes: as stand-alone executable, as command-line argument to **python**, interactively in the Python interpreter