2 Python

1991, Guido van Rossum, Math Center, Amsterdam
uses: interactive computing, scripting, prototyping, gluing code together
builtin functions, standard library, extensions
python vs. ipython
2.1 interactive computing

2.1.2 help
2.1.3 list

((DEMO))
from math import sin, cos
dir()  # directory of defined names
list:   [...]
       *
       +
       list(range(...))  list(map(...))
indexing  slicing  start:stop:stride
->  what is difference
  between indexing and slicing?
->  create a list of a thousand 'no's
#  slices
->  if l = list(range(5)), what is l[:,-1]?
->  instead of 1:6, we can use what?
2.1 interactive computing (contd)

2.1.4
\[ \text{str} \quad \# \quad \text{string} \]

2.1.5
containers
- sequences: list, str, tuple
  a tuple is an immutable list
- dict
  a key instead of an index
  (a key must be immutable)
- set
  a key but no value

2.1.6 operators
precedence
short circuit evaluation

((DEMO))
2.1.4–2.1.6 review

str: '...' * + %
    str.split(...) str.join(...)
-> convert a "float" to a "str"
    using up to 1000 digits
containers:
    sequences: list, str*
        tuple* (item, ...) *=immutable
dict    {key: item, ...}
del
2.2 developing a script

sys module
    argv, path

2.2.1 control structures
    if ...:      | for ... in ...:
        ...      | ...
    elif ...:
        ...      | while ...
    else:
        ...

    ...

    def     # define a function
2.2.2 Objects
object = unit of data residing in memory
the dot operator
   <module>..<identifier> sys.argv, math.sin
   <type>..<identifier>
       str.split, list.append
!! <object of that type>..<identifier>
   '12 31'.split
2.2.3 Reading and writing
open, <file_obj>.close, <file_obj>.write,
   <file_obj>.readline
sys.stdout, sys.stdin, sys.stderr
filter: reads from sys.stdin,
     writes to sys.stdout

(((DEMO)))
2.2 review

sys.argv

control structures
    if ...: ... elif ...: ... else: ...
    for ... in ...: ...
    while ...: ...
    if item in mylist:
        index = list.index(mylist, item)
    else:
        index = -1
2.2 review (contd)

```python
for i in [2, 3, 5]: print(i)
for i in {1:2, 2:3, 3:5}: print(i)

ij = enumerate([2, 3, 5])
    # iterator for a sequence of tuples, i.e.,
list(ij) -> [(0, 2), (1, 3), (2, 5)]
    # looping on a tuple of variables
for i, j in ij:
    print(i, j)

def myfunc(myargs):
    ...
    return myvalues  # required?
    # last statement?
```
2.2 review (contd)

2.2.2 Objects

object = unit of data residing in memory

the dot operator

<module>.<identifier> sys.argv, math.sin
<type>.<identifier> str.split, list.append
<object of that type>.<identifier>

e.g., 'Break me up'.split()

2.2.3 Reading and writing

open, file.close, file.write, file.readline

Q: How can you avoid readline?

sys.stdout, sys.stdin, sys.stderr

filter: reads from sys.stdin,
writes to sys.stdout

Q: print x ➞ ?????.write(x ???)

io.TextIOWrapper.readline(sys.stdin)

-> sys.stdin.readline()
2.3 variables & functions

2.3.1 Variables and data
variables do not have types; objects do

2.3.2 Assignment and copy
variable = reference to an object
\[ \text{object} \]
\[
\begin{array}{c}
[. ] \rightarrow [3.14|\text{ref\_ct}|. ] \rightarrow \text{float}
\end{array}
\]
loosely speaking, reference = pointer
(to be precise, a pointer is a reference for which addition/subtraction is defined)

IMPORTANT for mutable objects like lists

2.3.3 Functions

((DEMO))
2.3 review

variables do not have types; objects do

\texttt{variable} = reference to an object

\times \quad \text{object}

\texttt{[ . ]} \longrightarrow \texttt{[3.14|ref\_ct| . ]} \longrightarrow float

IMPORTANT

for mutable objects like lists and dicts
2.3 review (contd)

\[ a = [1] \times 3; \quad b = [2] \times 3 \]

Which of these are equivalent?

(i) \( a = b \)
(ii) \( a[:] = b \)
(iii) \( a = b[:] \)
(iv) \( a[:] = b[:] \)

If \( b \) is a list and \( b[:] \) is not on the LHS of =

\[
\begin{array}{c}
\text{b} \\
\text{slice(} \text{None)}
\end{array}
\]

\[ \Rightarrow \text{it is a newly constructed object} \]
2.3 review (contd)

Python uses call by ________
what does this mean?
<parm1> = <arg1>; <parm2> = <arg2>; ...

what is the return value
for list.append([2, 3], 5)?

when would a variable x in a function
be considered to be global?
(i) ... (ii) ...

DICTA:
1. NO side effects / no mutation of arguments
   except for the 1st argument of a _method_
i.e., a function of the form
   <type>.<identifier>
2. NO global variables
unit testing:
    if __name__ == '__main__':

Sec 2.4.1: exception handling
    try:
        ....
    except <exception> as <message>:
        <action>
also, you can
    raise <exception>(<message>)
Sec 2.4.2: assertions

‘‘A week of hard work
can sometimes save you an hour of thought.’’

Sec 2.4.3: side effects

Sec 2.4.4: debugging

Sec 2.4.5: coding standards

Sec 2.4.6: documentation

docstring

(DEMO)
unit testing:
    embed module in script
    if __name__ == '__main__':
Do not use import for development
__Sec 2.4.1: exception handling__
to handle an exception use
    ???:
        ...
    ??? ???, ???:
        ...
    ??? ???, ???:
    ...
to create an exception use
    ??? ???, ???
2.4 review (contd)

__Sec 2.4.2: assertions__
to detect a bug
??? ???, ???
(can be turned off with python −O)

__Sec 2.4.4: debugging__
ipython: run −d <script>
list break step c quit help disable
2.5 the os module

the os module

datum: sys.platform

functions: subprocess.call, glob.glob, tempfile.mktemp

((DEMO))
modules: os
functions subprocess.call, glob.glob, tempfile.mktemp
2.6 object-oriented programming

learning Python & C helpful for C++

defining new data types: classes

instructor has major misgivings about OOP
1. there is more for the reader to learn before he* can read the code
2. programmers have greater opportunity to be creative

avoid defining classes; explore alternatives the examples given here are not all intended to be emulated

the simplest class—-I call it struct-
constructors

a Polynomial class

special methods
  a.__add__(b) \iff a + b
  f.__call__(x) \iff f(x)
  a.__setitem__(i, ai) \iff a[i] = ai

the Polynomial class revisited

three principles:
  encapsulation
  inheritance
  polymorphism

a SparseVector class

((DEMO))
Example 1:

```python
class struct_(object):  # (object) is optional
    pass
parm = struct()
parm.tol = 1.e-6
parm.itmax = 50
```
special methods

\[ \text{\textless expr1\textgreater} + \text{\textless expr2\textgreater} \]

\[ \Longleftrightarrow \text{\textless expr1\textgreater}._{\text{\_\_add\_\_}}(\text{\textless expr\textgreater}) \]

except that the 2nd form
may need additional parentheses

special methods:

a._{\text{\_\_add\_\_}}(b)  # a + b
l._{\text{\_\_len\_\_}}()  # len(l)
f._{\text{\_\_call\_\_}}(\text{\textless args\textgreater})  # f(\text{\textless args\textgreater})
a._{\text{\_\_setitem\_\_}}(i, a_i)  # a[i] = a_i
<object>._{\text{\_\_init\_\_}}(\text{\textless args\textgreater})
    # called automatically
    # after calling \langle class \rangle(\langle args \rangle)
2.6 review (contd)

Example 2:

```python
class Polynomial(object):
    def __init__(self, coeff):
    def __call__(self, x):
    def __add__(self, other):
    def degree(self):

p = Polynomial([-2., 0., 1.])
print(p(0.5), p.degree())
```

three principles:
- encapsulation: access by methods only
- inheritance: subclasses
- polymorphism: overloading functions and operators