Python

CIS 218
Oakton Community College
## Python features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>no compiling or linking (see text)</td>
<td>rapid development cycle</td>
</tr>
<tr>
<td>no type declarations</td>
<td>simpler, shorter, more flexible</td>
</tr>
<tr>
<td>automatic memory management</td>
<td>garbage collection</td>
</tr>
<tr>
<td>high-level data types and operations</td>
<td>fast development</td>
</tr>
<tr>
<td>object-oriented programming</td>
<td>code structuring and reuse, C++</td>
</tr>
<tr>
<td>embedding and extending in C</td>
<td>mixed language systems</td>
</tr>
<tr>
<td>classes, modules, exceptions</td>
<td>&quot;programming-in-the-large&quot; support</td>
</tr>
<tr>
<td>dynamic loading of C modules</td>
<td>simplified extensions, smaller binaries</td>
</tr>
<tr>
<td>dynamic reloading of C modules</td>
<td>programs can be modified without stopping</td>
</tr>
</tbody>
</table>

CIS 218
## Python features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal &quot;first-class&quot; object model</td>
<td>fewer restrictions and rules</td>
</tr>
<tr>
<td>run-time program construction</td>
<td>handles unforeseen needs, end-user coding</td>
</tr>
<tr>
<td>interactive, dynamic nature</td>
<td>incremental development and testing</td>
</tr>
<tr>
<td>access to interpreter information</td>
<td>metaprogramming, introspective objects</td>
</tr>
<tr>
<td>wide portability</td>
<td>cross-platform programming without ports</td>
</tr>
<tr>
<td>compilation to portable byte-code</td>
<td>execution speed, protecting source code</td>
</tr>
<tr>
<td>built-in interfaces to external services</td>
<td>system tools, GUIs, persistence, databases, etc.</td>
</tr>
</tbody>
</table>
Python

- See:
  - https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_2.6/
  - https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3

- elements from C++, Modula-3 (modules), ABC, Icon (slicing)

- same syntax family as Perl, Tcl, Scheme, REXX, BASIC dialects
Python structure

- modules: Python source files or C extensions
  - import, top-level via from, reload

- statements
  - control flow
  - create objects
  - indentation matters – instead of {} 

- objects
  - everything is an object
  - automatically reclaimed when no longer needed
Uses of Python

- shell tools
  - system admin tools, command line programs
- extension-language work
- rapid prototyping and development
- language-based modules
  - instead of special-purpose parsers
- graphical user interfaces
- database access
- distributed programming
- Internet scripting
- Not as efficient as “C”
Python Syntax

- Everything is case sensitive
- Don’t mix tabs and spaces
- Python is object oriented
- Objects come from classes
- Object attributes are specified with a dot after the object name: object.attr
- An object attribute is also called a “method”
- See also “name spaces”
Using python

- `#!/usr/bin/python`
  - `print "Hello World"`

- interactive use
  - `python –c command [arg] ...`
  - `python –i script`
    - read script first, then interactive
Hello World

- python -c 'print "Hello World"
- python –i hello.py

- `#!/usr/bin/python
  print "Hello World"

- `#!/usr/bin/python
  print 'Hello World'

CIS 218
# stdin/stdout

- `#!/usr/bin/python`
  - `input = raw_input ( 'Enter your name: ' )`
  - `print 'hello' + input + '!'`
- `#!/usr/bin/python`
  - `print 'Enter you name:'`
  - `input = raw_input ( )`
  - `print 'hello' + input + '!'`
- `#!/usr/bin/python`
  - `input = raw_input ( 'Enter your age: ' )`
  - `age = int(input)`
  - `age = age + 1`
  - `print 'Next year you will be:' str(age)`

Note raw_input replaced with input in python V3
Python Data Types

- **boolean:**
  - True and False. Mostly interchangeable with the integers 1 and 0. `boolean False = integer 0` and the empty string "" as equivalent to False, and all other values as equivalent to True.

- **Numeric types:**
  - `int`: Integers; equivalent to C longs in Python 2.x, non-limited length in Python 3.x
  - `long`: Long integers of non-limited length; exists only in Python 2.x
  - `float`: Floating-Point numbers, equivalent to C doubles
  - `complex`: Complex Numbers
  - In general, the number types are automatically 'up cast' in this order:
    - Int → Long → Float → Complex. The farther to the right you go, the higher the precedence.

- **Sequences:**
  - `str`: String; represented as a sequence of 8-bit characters in Python 2.x, but as a sequence of Unicode characters (in the range of U+0000 - U+10FFFF) in Python 3.x
  - `byte`: a sequence of integers in the range of 0-255; only available in Python 3.x
  - `byte array`: like bytes, but mutable (see below); only available in Python 3.x
  - `List` – an indexed array
  - `Tuple` – immutable set

- **Sets:**
  - `set`: an unordered collection of unique objects; available as a standard type since Python 2.6
  - Frozen set: like set, but immutable (see below); available as a standard type since Python 2.6

- **Mappings:**
  - `dict`: Python dictionaries, also called hashmaps or associative arrays, which means that an element of the list is associated with a definition – i.e. value and unique key pair.
Mutable vs Immutable Objects

Data types in Python can be distinguished based on whether objects of the type are mutable or immutable. Immutable objects cannot be changed after they are created.

Some immutable types:
- int, float, long, complex
- str
- bytes
- tuple
- Frozen set

Some mutable types:
- byte array
- list
- set
- dict

Only mutable objects support methods that change the object in place, such as reassignment of a sequence slice, which will work for lists, but raise an error for tuples and strings.
Basic scalar operations

- Implicit “print”: var = “Hello World”
  - print var
  - var
  - “Hello World”
- Assignment:
  - size = 40
  - a = b = c = 3
- Numbers
  - integer, float
  - complex numbers: 1j+3, abs(z)
- Strings
  - 'hello world', 'it\'s hot'
  - "bye world"
  - continuation via \ or use """" long text """"
Scalar Built-in Functions

- `exit()` - return return code (and exits)
- `float()` - returns float
- `int()` - returns integer
- `len()` - returns length
- `map()` - function value return
- `max()` - returns max value in list
- `range()` - returns a range of values
- `sorted()` - ordered list
- `str()` - returns string
- `type()` - returns argument type
Math

- + add
- - subtract
- * multiply
- / divide
- ** exponent - OR – pow(x, y)
- % modulus (remainder) – OR – od divmod(x, y)
- abs()
- long()
- float()
- -()
- math.sqrt(a) – uses import math
Math examples

- $N = 0$
  
  $$N \ += \ 5$$

- $N1=5$

- $N2 = 8$

- $N1 \ + \ N2$

- $3/2$

- float(3)/2

- $2**3$
String operations

- There are three ways you can declare a string in Python: single quotes ('), double quotes (""), and triple quotes ("""") for long string operations.

- Concatenate with + or neighbors
  - `word = 'Help' + x`
  - `word = 'Help' 'a'`

- Subscripting of strings [as list starts with 0]
  - `'Hello'[2] → 'l'`
  - Slice: `'Hello'[1:2] → 'el'`
  - `word[-1] → last character`
  - `len(word) → 5`
  - Immutable: cannot assign to subscript
Regular Expressions

- Uses RE module – import re

- Methods:
  findall(regex, string)
  match(regex, string) .. Find first occurrence
  search(regex,string) returns MatchObject
  bool('string') – does it exist?
  group() – returns MatchObject value
  type () – type of MatchObject
Lists

- Sequential list in square brackets separated by commas. Index starts at 0

- Lists can be heterogeneous
  - \[ a = ['spam', 'eggs', 100, 1234, 2*2] \]

- Lists can be indexed and sliced:
  - \[ a[0] \rightarrow \text{spam} \]
  - \[ a[:2] \rightarrow ['spam', 'eggs'] \]

- Lists can be manipulated
  - \[ a[0:2] = [1, 12] \]
  - \[ a[0:0] = [] \]
  - \[ \text{len}(a) \rightarrow 5 \]
List xamples

- `a = [ 'bb', 'dd', 'zz', 'rr' ]`
- `a[2]` .. List scalar
- `a[1] = 'qqqq'` .. Reassign value
- `x=a[0]` .. reassign
- `a[-1], a[-2]` .. Index from end
- `a[:2], a[2:], a[1:-1]` .. : specifies list end or begin
- `b = a` .. Reassign list by reference
- `b = a[:]` .. Copy without reference
List methods

- `append(x)`
- `extend(L)`
  - append all items in list (like Tcl `lappend`)
- `insert(i, x)`
- `remove(x)`
- `pop([i]), pop()`
  - create stack (FIFO), or queue (LIFO) → pop(0)
- `index(x)`
  - return the index for value `x`
List methods

- **count(x)**
  - how many times x appears in list

- **sort()**
  - sort items in place

- **reverse()**
  - reverse list
del - removing list items

- remove by index, not value
- remove slices from list (rather than by assigning an empty list)

```python
>>> a = [-1, 1, 66.6, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.6, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.6, 1234.5]
```
Tuples

- Declared and handled same way as list. But are immutable – e.g. cannot be changed once declared.

- Tuples use the same REFERENCE and OUTPUT operations as list, but not updates.
Sets

- Sets are just like lists, except that they are unordered and they do not allow duplicate values.

- Elements of a set are neither bound to a number (like list and tuple) nor to a key (like dictionary).

- Used for faster access to a large # of items

- Can be derived from lists or tuples.
Dictionary

- Keyed list; e.g. Perl "hash"
- Set up as dict = {'key': value, 'key': value, .... }
- Keys/values need not be of a consistent type – heterogenous
- like Perl Tcl or awk associative arrays
- indexed by keys
- keys are any immutable type: e.g., tuples
- but not lists (mutable!)
- no particular order
- delete elements with del

```python
>>> del tel['foo']
```
- keys() method → unsorted list of keys

```python
>>> tel.keys()
['cs', 'lennox', 'hgs']
```
- use has_key() to check for value existence

```python
>>> tel.has_key('foo')
```
Dictionary Examples

- Examples
- ext = {'sam': 44, 'max': 88, 'zach': 22}
- ext['helen'] = 92
- ext['max'] = 150
- del ext['max']
- ext['zach']
- ext = {'sam': 44, 'max': 88, 'zach': 22}
  for key in ext:
    print key, ext[key]
  print "max" in ext
Dictionary methods

- items – key value pairs
- keys – key list
- values – value list
- del – remove value/key pair
- has_key - if value exists
Conditions

- Placed in parentheses like C. Uses BASH condition checking.
- can check for sequence membership with is and is not:
  ```python
global vec = [1, 2, 3, 4, 5]
>>> if (4 in vec):
...  print '4 is'
4 is
```  
- chained comparisons: a less than b AND b equals c:
  ```python
a < b == c
```  
- Can assign comparison to variable:
  ```python
>>> s1,s2,s3='','foo', 'bar'
>>> non_null = s1 or s2 or s3
>>> non_null
foo
```  
- AND and OR are short-circuit operators:
  - evaluated from left to right
  - stop evaluation as soon as outcome clear
Conditions

- Operator - function
- < - less than
- <= - less than or equal to
- > - greater than
- >= - greater than or equal to
- == - equal
- != - not equal
- is / is not – test object identity
- No direct file checking – uses a separate library
Comparing sequences

- can compare sequences (lists, tuples, scalars ...)
- lexicographical comparison:
  - compare first; if different → outcome
  - continue recursively
  - subsequences are smaller
  - strings use ASCII comparison
  - can compare objects of different type, but by type name (list < string < tuple)
  - can compare sequences (lists, tuples, scalars ...)

CIS 218
Comparing sequence examples

(1,2,3) < (1,2,4)

[1,2,3] < [1,2,4]

'ABC' < 'C' < 'Pascal' < 'Python'

(1,2,3) == (1.0,2.0,3.0)

(1,2) < (1,2,-1)
Control structure: - while

Basic control syntax:
Conditions in parentheses (recommended),
terminated by colon, code block is indented

a,b = 0, 1
# non-zero = true
while (b < 10):
    # formatted output, without \n
    print b,
    # multiple assignment
    a,b = b, a+b

No until
Control structure: if

```python
x = int(raw_input("Please enter #:"))
if (x < 0):
    x = 0
    print 'Negative changed to zero'
elif (x == 0):
    print 'Zero'
elif (x == 1):
    print 'Single'
else:
    print 'More'
```

- no case statement
Control structure: for

```python
a = ['cat', 'window', 'defenestrate']
for x in a:
    print x, len(x)
```

- no arithmetic progression, but
  - `range(10) → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
  - `for i in range(len(a)):`
    ```python
    print i, a[i]
    ```
- do not modify the sequence being iterated over
Loops: break, continue, else, pass

- **pass** does nothing

\[
\text{while 1:}
\]
\[
\text{    pass}
\]

- **break** and **continue** like C

- **else** after loop exhaustion

```python
for n in range(2, 10):
    for x in range(2, n):
        if (n % x) == 0:
            print n, 'equals', x, '*', n//x
            break
        else:
            # loop fell through without finding a factor
            print n, 'is prime'
```

CIS 218
Functions

- `def` functionname (arg1, arg2, ...):
- statement1
- statement2
- return value  ... can also use print
- Arguments can be assigned default values in the function be specifying (arg=value) in the argument list
- Arguments are passed by reference, meaning you point to the original item.
- Mutable objects can be changed by a function.
- Variables are local unless defined globally.
- Functions can be nested
Defining functions

def fib(n):
    """Print a Fibonacci series up to n."""
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b
    return b

answer = fib(2000)

- First line is mandatory
- first look for variables in local, then global
- need global to assign global variables
Functions: default argument values

def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while 1:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'): return 1
        if ok in ('n', 'no'): return 0
        retries = retries - 1
        if retries < 0: raise IOError, 'refusenik error'
        print complaint

>>> ask_ok('Really?')
Keyword arguments

- last arguments can be given as keywords

```python
def parrot(voltage, state='a stiff', action='voom',
          type='Norwegian blue'):
    print "-- This parrot wouldn't", action,
    print "if you put", voltage, "Volts through it."
    print "Lovely plumage, the ", type
    print "-- It's", state, "!"

parrot(1000)
parrot(action='VOOOM', voltage=100000)
```
Functional programming tools

- **filter**(function, sequence)
  ```python
def f(x): return x%2 != 0 and x%3 0
filter(f, range(2,25))
```

- **map**(function, sequence)
  - call function for each item
  - return list of return values

- **reduce**(function, sequence)
  - return a single value
  - call binary function on the first two items
  - then on the result and next item
  - iterate
Modules

- collection of functions and variables, typically in scripts
- definitions can be imported – import module
- file name is module name + .py
- **import** math .... Standard math module
  
  ```python
  print math.sqrt(10)
  ```
- Can create own modules – e.g. module *fibo.py*
  ```python
def fib(n):   # write Fib. series up to n   ...
def fib2(n): # return Fib. series up to n
  ```
- function definition + executable statements
- executed only when module is imported
- modules have private symbol tables
- avoids name clash for global variables
- accessible as *module.globalname*
- can import into name space:
  ```python
  >>> from fibo import fib, fib2
  >>> fib(500)
  ```
- can import all names defined by module:
  ```python
  >>> from fibo import *
  ```
Module search path

- current directory
- list of directories specified in PYTHONPATH environment variable
- uses installation-default if not defined, e.g., ./usr/local/lib/python
- uses sys.path
  ```
  >>> import sys
  >>> sys.path
  ['', 'C:\PROGRA~1\Python2.2', 'C:\Program Files\Python2.2\DLLs', 'C:\Program Files\Python2.2\lib', 'C:\Program Files\Python2.2\lib\lib-tk', 'C:\Program Files\Python2.2\site-packages']
  ```
Standard library modules

- A list of the Standard Library modules can be found at [http://www.python.org/doc/](http://www.python.org/doc/). The following are among the most important:
  - time
  - sys
  - os
  - math
  - random
  - pickle
  - urllib
  - re
  - cgi
  - socket
Module listing

- use `dir()` for each module

```python
>>> dir(fibo)
['__name__', 'fib', 'fib2']

>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__stderr__', '__stdin__', '__stdout__', '_getframe', 'argv', 'builtin_module_names', 'byteorder', 'copyright', 'displayhook', 'dllhandle', 'exc_info', 'exc_type', 'excepthook', 'exec_prefix', 'executable', 'exit', 'getdefaultencoding', 'getrecursionlimit', 'getrefcount', 'hexversion', 'last_type', 'last_value', 'maxint', 'maxunicode', 'modules', 'path', 'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'version', 'version_info', 'warnoptions', 'winver']
```
Compiled Python files

- include byte-compiled version of module if there exists fibo.pyc in same directory as fibo.py
- only if creation time of fibo.pyc matches fibo.py
- automatically write compiled file, if possible
- platform independent
- doesn't run any faster, but loads faster
- can have only .pyc file → hide source
Classes

- classes (and data types) are objects
- built-in types cannot be used as base classes by user
- arithmetic operators, subscripting can be redefined for class instances (like C++, unlike Java)
Namespaces

- mapping from name to object:
  - built-in names (\texttt{abs()})
  - global names in module
  - local names in function invocation
- attributes = any following a dot
  - \texttt{z.real}, \texttt{z.imag}
- attributes read-only or writable
  - module attributes are writeable
Namespaces

- scope = textual region of Python program where a namespace is directly accessible (without dot)
  - innermost scope (first) = local names
  - middle scope = current module's global names
  - outermost scope (last) = built-in names
- assignments always affect innermost scope
  - don't copy, just create name bindings to objects
- global indicates name is in global scope
Method objects

- Called immediately:
  \[x.f()\]
- can be referenced:
  \[xf = x.f\]
  
  \[while 1:\]
  \>
    \[print xf()\]
- object is passed as first argument of function \[\rightarrow\] 'self'
  - \[x.f()\] is equivalent to MyClass.f(x)
File I/O

- `open (‘file pathname’, ‘mode’)`
- Mode: `r` – read, `w` – write, `r+` - `R=W`, `a` – append, `a+` append and read, `b` – binary
- Methods `close()`, `isatty()`, `read()` – until `EOF`, `readline()`, `readlines()`, `write(string)`, `writelines(stringlist)`

CIS 218
Exceptions

- syntax (parsing) errors

```python
while 1 print 'Hello World'
```

File "<stdin>"., line 1
  while 1 print 'Hello World'
    ^

SyntaxError: invalid syntax

- exceptions
  - run-time errors
  - e.g., `ZeroDivisionError`, `NameError`, `TypeError`
Handling exceptions

```python
while 1:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print "Not a valid number"
```

- First, execute `try` clause
- if no exception, skip `except` clause
- if exception, skip rest of `try` clause and use `except` clause
- if no matching exception, attempt outer `try` statement
Handling exceptions

- **try.py**

```python
import sys
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print 'cannot open', arg
    else:
        print arg, 'lines:', len(f.readlines())
    f.close
```

- e.g., as `python try.py *.py`
## Language comparison

<table>
<thead>
<tr>
<th></th>
<th>Tcl</th>
<th>Perl</th>
<th>Python</th>
<th>JavaScript</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>development</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>regexp</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>breadth</strong></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>extensible</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>embeddable</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>easy GUI</td>
<td>✓</td>
<td></td>
<td>✓ (Tk)</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>net/web</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>enterprise</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cross-platform</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I18N</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread-safe</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>database access</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>