Python

Introduction

- Developed by Guido von Rossum in 1991
- Successor to ABC programming language
- Version 2.0 released in 2000
  - Garbage collection & Unicode support
- Version 3.0 released in 2008
  - Revised I/O (Version 3.0 backward/forward incompatible w/ Version 2.0)

Application of Python

- Writing basic CGI scripts
- Bioinformatics
- Physics
- Commercial game development
- 3D graphics rendering
- Programming language of choice for Raspberry Pi

Creating/Interpreting Python file

- Creation:
  - Simple editor (Notepad, Notepad++, Wordpad)
  - Commonly use “.py” as file extension (eg: “myscript.py”)
- Interpreting:
  - “python myscript.py”
  - If errors/warnings, message(s) displayed
  - If no errors, file is executed

Simple Python program – “Hello World!”

- Program:

  print (“Hello World”)

- Explanation:

1. The string must be enclosed in quotes and parentheses ("...") or (‘…’)
2. {Note: the end-of-line character is implicit; for additional lines, use “\n”}
3. {Note: to avoid a default newline, use: end=“”
   ex: print (“Hello World”, end=“”)
4. {Note: there is no “;” to end a statement}

Variables and Assignments

- Example:

```
width = 20
height = 5*9
print (width * height)
```

- Output: 900
- Can assign a value to several variables simultaneously: x = y = z = 0
- Variables must be assigned a value before being used
- Full support for floating point: 7.0 / 2 = 3.5
- Complex numbers: x=3+2j OR x=complex (3,2)
  - real component: x.real, imaginary component: x.imag

Strings

- Format (either “…” or ‘…’):

```
print (“Hello there”)
print (‘Hello there’)
```

- Multiple lines:

```
print (“””
        Line 1
        Line 2
        Line 3
“””)
```

String Concatenation / Repetition

- Concatenation:

```
greeting = ‘Hello’ + ‘world’
greeting = ‘Hello ’ ‘world’
```

- Repetition:

```
print (greeting*3)
```

Output: Hello world Hello world Hello world

String indices / Length

- Example:

```
x = “Hello world!”
print (x[4])              # prints “o”
print (x[0:3])            #prints “Hel”, range 0…(3-1)
print (x[:4])             #prints “Hello”
print (x[6:])             #prints “world!”
print (x[-2:-1])          #prints “d”, range -2…((-1)-1)
print (len(x))            #prints string length of 12
```

- Degenerate (out of bounds) indices:

```
print (x[1:100])          #prints “ello world!”
print (x[100:])           #prints “”
```

String Substitution

- Format: string.replace(old, new, max)
  {max: optional maximum number of replacements}

- Example:

```
str = “I love CIT 210!”
str = str.replace(“210”, “270”)
print (str)
```

- Output:

```
I love CIT 270!
```

Lists

- Format:

```
list = [item1, item2, …]
```

- Example:

```
my_list = [‘cit’, ‘jd1538’, 270]
```

{can be mixed types}

- Indices:

```
print (my_list[0])            # prints “cit”
print (my_list[-1])           # prints “270”
print (my_list[0] + ‘210’)    # prints “cit210”
print (my_list[0]*3)          # prints “citcitcit”
print (my_list[2]-60)         # prints “210”
```

Lists (cont.)

- Length:

```
a = [‘a’, ‘b’, ‘c’, ‘d’]
print (len(a))          # prints “4”
```

- Nested lists:

```
q = [2,3]
p = [1,q,4]
print (p[1])            # prints [2,3]
print (p[1][0])         # prints [2]
```

While Loop

- Format (all statements within loop are indented):

```
while (condition_is_true):
        statements
```

- Example:

```
# Fibonacci series
# the sum of two elements defines the next
a, b = 0, 1
while b < 10:
        print (b, end=“ ”)
        a, b = b, a+b
```

Input

- Format (for strings):

```
variable = input(“String”)
```

- Format (for integers):

```
variable = int(input(“String“))
```

- Example:

```
x = int(input("Please enter an integer: "))
```

If Condition

- Format (all statements within loop are indented):

```
if (condition_is_true):
        statements
elif (condition_is_true):
        statements
…
else:
        statements
```

If Condition (example)

```
x = int(input("Please enter an integer: "))
if x < 0:
        x = 0
        print ('Negative changed to zero‘)
elif x == 0:
        print ('Zero‘)
elif x == 1:
        print ('Single‘)
else:
        print ('More‘)
```

For Loop

- Format:

```
for each_element in list:
        statements
```

- Example:

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

- Output:

```
cat 3
window 6
comp_info_technology 20
```

Range() function

- Example:

```
range(10) = [0,1,2,3,4,5,6,7,8,9]       #[0, max-1]
range(5,10) = [5,6,7,8,9]               #[min, max-1]
range(0,12,3) = [0,3,6,9]        #[min, max-1, step]
```

- Range in for-loop:

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
        print (i, a[i])
```

Splitting a String

- Split a string into a list of strings

- Format: list = string.split(deliminator)

- Example:

  ```
  my_string = “I love CIT 270!”
  my_list = my_string.split(“ “)

  for i in range(len(my_list)):
          print my_list[i]
  ```

- Output:

  ```
  I
  love
  CIT
  270!
  ```

break & continue & else

- break:
  - Breaks out of closest enclosing for-loop
- continue:
  - Continues with next iteration of loop
- else:
  - Executes when loop terminates w/o iterating at least once

break,else Example

Example:

```
for n in range(2, 6):
        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 a prime number‘)
```

Output:

```
   2 is a prime number
   3 is a prime number
   4 equals 2 * 2
   5 is a prime number
```

pass Statement

- Does nothing, used when statement required
- Example:

```
while True:
    pass            #wait for keyboard interrupt
```

- Can be used as place-holder for function or conditional body

Fancy Output Formatting

- Convert value to string:

  str(n) returns representation of value n which is human-readable

  repr(n) generates representation of value for reading by the interpreter
- Conversion functions are same for numbers, lists, & dictionaries
- Conversion functions may differ for floating point numbers and strings

Fancy Output Formatting Example (Interactive Python mode)

```
>>> str(1.0/7.0)
'0.142857142857'
>>> repr(1.0/7.0)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is '
+ repr(y) + '...'
>>> print (s)
The value of x is 32.5, and y is 40000...
>>> # The repr() of a string adds string quotes and
backslashes:
... hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print (hellos)
'hello, world\n'
```

Formatting for Precision

Format:

```
print (‘%(#).2f’ %{“#”:x})
```

{where x is value to be printed}

Example:

```
>>> x=(1.0/7.0)
>>> print (x)
0.142857142857
>>> print (‘%(#).2f’ %{“#”:x})
0.14
```

Left/Right Justification

- Left justification of value x within n spaces:

  ```
  repr(x).ljust(n)
  ```

- Right justification of value x within n spaces:

  ```
  repr(x).rjust(n)
  ```

- Center a value x within n spaces:

  ```
  repr(x).center(n)
  ```

- Pad a numeric string x with n zeros

  ```
  x.zfill(n)
  ```

Right Justification Example

```
>>> for x in range(1, 11):
...     print (repr(x).rjust(2),repr(x*x).rjust(3),
repr(x*x*x).rjust(4))
…
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

Functions

- Format for defining a function:

```
def function_name(parameters):
        “””optional function description”””
        statements
```

- Example:

```
def fib(n):    # write Fibonacci series up to n
        """Print a Fibonacci series up to n."""
        a, b = 0, 1
        while a < n:
                print (a, end=“ “)
                a, b = b, a+b

#function call
fib(2000)
```

Return statement in Function

```
def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)    # see below
        a, b = b, a+b
    return result

f100 = fib2(100)        # call it
f100                    # write the result
```

Functions w/ varying arguments

- Allows function to be called w/ less arguments than defined to allow

- Example:

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    …

#Valid calls to the function
ask_ok('Do you really want to quit?')
ask_ok('OK to overwrite the file?', 2)
ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')
```

Keyword Arguments

- Example:

```
def node(voltage, state=‘active', action=‘load', type=‘empircal'):
        print ("The device would not", action, end=“ “)
        print ("if you applied", voltage, "volts through it.“, end= “ “)
        print (" Since the units are", type, end=“ “)
        print (“the device should be”, state, "!“, end= “ “)
```

- Output:

The device would not load if you applied ______ volts through it. Since the units are empirical the device should be active!

Keyword Arguments (cont.)

- Example:

```
def node(voltage, state=‘active', action=‘load', type=‘empircal'):
```

- Valid calls:

```
node(1000)                              # 1 positional argument
node(voltage=1000)                      # 1 keyword argument
node(voltage=1000, action=‘store')      # 2 keyword arguments
node(action=‘store', voltage=1000)      # 2 keyword arguments
node(1000, ‘wait', ‘restore')           # 3 positional arguments
node(1000, state=‘sleep')               # 1 positional, 1 keyword
```

Keyword Arguments (cont.)

- Example:

```
def node(voltage, state=‘active', action=‘load', type=‘empircal'):
```

- Invalid calls:

```
node()                          # required argument missing
node(voltage=5.0, 'dead')       # non-keyword argument after a keyword argument
node(110, voltage=220)          # duplicate value for the same argument
node(actor=‘Will Smith')        # unknown keyword argument
```

Document Strings

- Example:

```
def my_function():
        """Do nothing, but document it.

          No, really, it doesn't do anything.
        """
        pass

print my_function.__doc__
```

- Output:

```
Do nothing, but document it.

No, really, it doesn't do anything.
```

List methods

- list.append(*x*)
  - Add an item to the end of the list; equivalent to a[len(a):] = [x].
- list.extend(*L*)
  - Extend the list by appending all the items in the given list; equivalent toa[len(a):] = L.
- list.insert(*i*, *x*)
  - Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).
- list.remove(*x*)
  - Remove the first item from the list whose value is *x*. It is an error if there is no such item.

List methods (cont.)

- list.pop([*i*])
  - Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list
- list.index(*x*)
  - Return the index in the list of the first item whose value is *x*. It is an error if there is no such item.
- list.count(*x*)
  - Return the number of times *x* appears in the list.
- list.sort()
  - Sort the items of the list, in place.
- list.reverse()
  - Reverse the elements of the list, in place.

List methods Example (Interactive mode)

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print (a.count(333), a.count(66.25), a.count('x'))
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Using Lists as Stacks Example

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

del() function

- Remove item from list with the specified index
- Example:

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
>>> del a
>>> a                 # will result in an error
```

Sets

- Unordered collection w/ no duplicate elements
- Basic uses:
  - Membership testing
  - Eliminating duplicate entries
- Operations:
  - Union, intersection, difference, symmetric difference

Sets Example

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)           # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit            # fast membership testing
True
>>> 'crabgrass' in fruit
False
```

Sets Example (cont.)

```
>>> # Demonstrate set operations on unique letters from two words

>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                              # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                              # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                              # letters in both a and b
set(['a', 'c'])
>>> a ^ b                              # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

Dictionaries

- Indexed by keys like hashes
- Keys can be either strings or numbers
- Also known as:
  - Associative memory or associate array
- Useful functions:
  - keys(n) returns list of all keys in dictionary n
  - sorted (n) sorts keys in dictionary n
  - del n[‘key’] deletes key-value pair from n

Dictionary Example

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Looping through Dictionaries: items() method

- Key & corresponding value can be retrieved at same time
- Example:

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print (k, v)
...
gallahad the pure
robin the brave
```

Looping through Dictionaries: enumerate() method

- Loop through sequence, retrieve position index and corresponding value at same time
- Example:

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print (i, v)
...
0 tic
1 tac
2 toe
```

Comparing Sequences

- Lexicographical ordering (ASCII, numerical ordering)
- Example:

```
(1, 2, 3)              < (1, 2, 4)
[1, 2, 3]              < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4)           < (1, 2, 4)
(1, 2)                 < (1, 2, -1)
(1, 2, 3)             == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab'))   < (1, 2, ('abc', 'a'), 4)
```

Module

- File containing Python definitions and statements
- Module’s name available as value of global variable __name__

  {2 underscrores before and 2 after name}

Module Example (File: fibo.py)

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Module Example (Importing)

```
>>> import fibo

>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo‘
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Opening a File

- Format:

  open(filename, mode)

- Modes: ‘r’ (read), ‘w’ (write), ‘a’ (append), ‘r+’ (read and write)
- Example:

```
>>> f = open('/tmp/workfile', 'w')
>>> print (f)
<open file '/tmp/workfile', mode 'w' at 80a0960>
```

Reading a File

- Format (note: “size” argument is optional):

  f.read(size)   # read “size” bytes of file f

- Example:

```
>>> f.read()
'This is the entire file.\n‘
>>> f.read()
‘ ‘
```

Reading a File (cont.)

- Format:

  f.readline()   # read a single line of file f

- Example:

```
>>> f.readline()
'This is the first line of the file.\n‘
>>> f.readline()
'Second line of the file\n‘
>>> f.readline()
‘ ‘
```

Reading a File (cont.)

- Format:

  f.readlines()   # return list containing all lines of data in file

- Example:

```
    >>> f.readlines()
    ['This is the first line of the
file.\n', 'Second
    line of the file\n']
```

Writing a File

- Format:

  f.write(string)   # write “string” to file f

- Example:

```
    >>> f.write('This is a test\n')
    # convert number to string before
writing
    >>> value = ('the answer', 42)
    >>> s = str(value)
    >>> f.write(s)
```

Other File methods

- f.tell():
  - Return integer indicating current position in file, measured in bytes
- f.seek(offset, from_what):
  - Change file object’s position to “offset” from “from_what” argument (0=beginning, 1=current position, 2=end of file)
- f.close():
  - Close the file and free system resources used