Python Programming

Harry S. Delugach, Ph.D.
Computer Science Department
University of Alabama in Huntsville
SRE-RAM
November 9, 2017

A simpler language

- C:

```
printf("Int val is %d; string val is %-20s\n",
12, "hello");
```

- Python:

```
print("Int val is",12,"; string val is", "hello")
```

An interpreted language

- Statements are processed one-by-one
- Objects are created when they are assigned a value
- Interactive environment invoked to run python programs
  - Command line interface
  - IDLE runs a python shell
- Prompt is ">>> "
  - Continuation is "…"

Features

- Interpreted language with dynamic strong typing
- Simple but powerful data structures
- Supports object-oriented programming
- Integrated help system
- Integrated module (library) system
- Integrated documentation features
- Optional (and default-able) function arguments
- Code can be "compiled" for faster execution

Brief history

- Created in the early 1990's by Guido van Rossum at Stichting Mathematisch Centrum (Netherlands)
- 1995: Moved to Corporation for National Research Initiatives (CNRI) in Reston, Virginia
- 2000-2001: Moved to Python Software Foundation (PSF). Zope Corporation is a sponsoring member.
- All Python releases are Open Source
- Python versions 2.x.x were to end in 2015, but extended to 2020
- Python versions 3.x.x are backward-incompatible with 2.x.x

Dynamic typing

- Objects get a type when they get a value
- You can "ask" an object what type it is:

```
>>> a = 42
>>> type( a )
<class 'int'>

>>> a = 42.5
>>> type( a )
<class 'float'>

>>> a = "hello"
>>> type( a )
<class 'str'>
```

Strong typing

- Dynamic typing doesn't mean "anything goes"
  - Object reference must already exist
  - Objects' types must match

```
>>> a = 42
>>> b = 13
>>> a + b
55

>>> a = "hello"
>>> a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int

>>> fib("hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in fib
TypeError: '<' not supported between instances of 'int' and
'str'
```

Test an object's type

- You can test for an object's type:

```
>>> a = "hello"

>>> type(a)
<class 'str'>

>>> type(a) is str
True
```

Functions

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Some more features

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

Multiple assignment on one line

Block begin statement ends with ":"

No parentheses for conditions

Optional named arguments

Optional arguments

Multiple assignment evaluates all right-hand side before executing

Indentation denotes nesting

Default arguments

```
def ask_ok(prompt, retries=4, reminder='Please try again!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
        print(reminder)


>>> ask_ok( "Enter yes or no: ")
Enter yes or no: huh?
Please try again!
Enter yes or no: yes
True

>>> ask_ok( "Enter yes or no: ", reminder="Wrong answer!" )
Enter yes or no: huh?
Wrong answer!
Enter yes or no: yes
True
```

Conditional statement

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

Compound values

- **List** is an ordered set of values delimited by square brackets
  `[ "this", "is", "a", "test" ]`
- List can be heterogeneous
  `[ "some text", 42, 8.333, 'hello' ]`
- List can be empty
  `[ ]`
- List elements accessible by index in the usual way

```
>>> a = [ "this", "is", "a", "test" ]
>>> a[1]
'is'
>>> a[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

- List elements also accessible by "slicing"

Loop statement

"for" statement operates on a set of values in order:

```
>>> words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

Iterating with integers

- The "range" iterator produces a list of integers:

range(5) produces 0,1,2,3,4

range(2,5) produces 2,3,4

range(0,10,2) produces 0, 2, 4, 6, 8

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

List "slicing"

```
days =
  ["Sun","Mon","Tue","Wed","Thu","Fri","Sat" ]

>>> days[0:3]
['Sun', 'Mon', 'Tue']
>>> days[:2]
['Sun', 'Mon']
>>> days[2:]
['Tue', 'Wed', 'Thu', 'Fri', 'Sat']
>>> days[-1]
'Sat'
```

List functions

- append
- reverse
- sort
- len
- list "comprehensions"
  `[x**2 for x in range(10)]`

  produces
  `[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

Associative arrays

- Allows array "index" to be any key you want to look up a value

```
>>> 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}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

Function docstring

- Functions support a docstring (one or more lines)
  - Accessible through __doc__ data attribute

```
>>> def my_function():
...     """Doesn't do anything."""
...     pass      # a "do nothing" stmt
...

>>> print( my_function.__doc__)
Doesn't do anything.

>>> help(my_function)
Help on function my_function in module __main__:

my_function()
    Doesn't do anything.
```

Function annotations

- Function argument types and return type can be declared (but not checked!)

```
def square( n : float ) -> float:
  """Returns the square of the given argument."""
  return n * n

>>> square( 3.0 )
9.0
>>> square.__annotations__
{'n': <class 'float'>, 'return': <class 'float'>}
>> square.__doc__
'Returns the square of the given argument.'
```

Classes

- Python supports dynamically created classes
  - Constructor is called __init__
  - Self reference is "self" (equiv. to "this")

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Larger scale programming

- Use interactive shell for testing and debugging
- Put tested/debugged functions into files
- All functions in a single file are called a **module**
  - Modules included with **import** command
  - Modules can have executable statements for initialization the first time a module is loaded
- Python has many built-in modules that do not require **import**

Resources

- python.org
  - Download Python 3
  - Tutorials and Documentation