Python Primer 1: Types and Operators

The Python Interpreter

- Python is an interpreted language.
- Commands are executed through the Python interpreter.
  - The interpreter receives a command, evaluates that command, and reports the result of the command.
- A programmer defines a series of commands in advance and saves those commands in a text file known as source code or a script.
- For Python, source code is conventionally stored in a file named with the **.py** suffix (e.g., **demo.py**).

An Example Program

```
print('Welcome to the GPA calculator.')
print('Please enter all your letter grades, one per line.')
print('Enter a blank line to designate the end.')
# map from letter grade to point value
points = {'A+':4.0, 'A':4.0, 'A-':3.67, 'B+':3.33, 'B':3.0, 'B-':2.67,
          'C+':2.33, 'C':2.0, 'C':1.67, 'D+':1.33, 'D':1.0, 'F':0.0}
num_courses = 0
total_points = 0
done = False
while not done:
  grade = input( )                          # read line from user
  if grade == '':                           # empty line was entered
    done = True
  elif grade not in points:                 # unrecognized grade entered
    print("Unknown grade '{0}' being ignored".format(grade))
  else:
    num_courses += 1
    total_points += points[grade]
if num_courses > 0:                         # avoid division by zero
  print('Your GPA is {0:.3}'.format(total_points / num_courses))
```

Objects in Python

- Python is an object-oriented language and classes form the basis for all data types.
- Python's built-in classes:
  - the **int** class for integers,
  - the **float** class for floating-point values,
  - the **str** class for character strings.

Identifiers, Objects, and the Assignment Statement

- The most important of all Python commands is an assignment statement:

  temperature = 98.6

  - This command establishes temperature as an identifier (also known as a name), and then associates it with the object expressed on the right-hand side of the equal sign, in this case a floating-point object with value 98.6.

Identifiers

- Identifiers in Python are case-sensitive, so temperature and Temperature are distinct names.
- Identifiers can be composed of almost any combination of letters, numerals, and underscore characters.
- An identifier cannot begin with a numeral and that there are 33 specially reserved words that cannot be used as identifiers:

Reserved Words								
False	as	continue	else	from	in	not	return	yield
None	assert	def	except	global	is	or	try	
True	break	del	finally	if	lambda	pass	while	
and	class	elif	for	import	nonlocal	raise	with	

Types

- Python is a **dynamically typed** language, as there is no advance declaration associating an identifier with a particular data type.
- An identifier can be associated with any type of object, and it can later be reassigned to another object of the same (or different) type.
- Although an identifier has no declared type, the object to which it refers has a definite type. In our first example, the characters 98.6 are recognized as a floating-point literal, and thus the identifier **temperature** is associated with an instance of the float class having that value.

Objects

- The process of creating a new instance of a class is known as **instantiation**.
- To instantiate an object we usually invoke the constructor of a class:

  w = Widget()

  - This is assuming that the constructor does not require any parameters.
- If the constructor does require parameters, we might use a syntax such as

  w = Widget(a, b, c)
- Many of Python's built-in classes a **literal form** for designating new instances. For example, the command

  temperature = 98.6

  results in the creation of a new instance of the float class.

Calling Methods

- Python supports functions a syntax such as sorted(data), in which case data is a parameter sent to the function.
- Python's classes may also define one or more methods (also known as member functions), which are invoked on a specific instance of a class using the dot (".") operator.
- For example, Python's list class has a method named sort that can be invoked with a syntax such as data.sort( ).
  - This particular method rearranges the contents of the list so that they are sorted.

Built-In Classes

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of **set** class	✓
dict	associative mapping (aka dictionary)	

- A class is *immutable* if each object of that class has a fixed value upon instantiation that cannot subsequently be changed. For example, the **float** class is immutable.

The bool Class

- The **bool** class is used for logical (Boolean) values, and the only two instances of that class are expressed as the literals:

  True and False

- The default constructor, bool( ), returns False.
- Python allows the creation of a Boolean value from a nonboolean type using the syntax bool(foo) for value foo. The interpretation depends upon the type of the parameter.
  - Numbers evaluate to False if zero, and True if nonzero.
  - Sequences and other container types, such as strings and lists, evaluate to False if empty and True if nonempty.

The int Class

- The **int** class is designed to represent integer values with arbitrary magnitude.
  - Python automatically chooses the internal representation for an integer based upon the magnitude of its value.
- The integer constructor, int( ), returns 0 by default.
- This constructor can also construct an integer value based upon an existing value of another type.
  - For example, if f represents a floating-point value, the syntax int(f) produces the truncated value of f. For example, int(3.14) produces the value 3, while int(−3.9) produces the value −3.
  - The constructor can also be used to parse a string that represents an integer. For example, the expression int(137) produces the integer value 137.

The float Class

- The **float** class is the floating-point type in Python.
  - The floating-point equivalent of an integral number, 2, can be expressed directly as 2.0.
  - One other form of literal for floating-point values uses scientific notation. For example, the literal 6.022e23 represents the mathematical value 6.022×10^{23}.
- The constructor float( ) returns 0.0.
- When given a parameter, the constructor, float, returns the equivalent floating-point value.
  - float(2) returns the floating-point value 2.0
  - float('3.14') returns 3.14

The list Class

- A **list** instance stores a sequence of objects, that is, a sequence of references (or pointers) to objects in the list.
- Elements of a list may be arbitrary objects (including the None object).
- Lists are array-based sequences and a list of length n has elements indexed from 0 to n−1 inclusive.
- Lists have the ability to dynamically expand and contract their capacities as needed.
- Python uses the characters [ ] as delimiters for a list literal.
  - [ ] is an empty list.
  - [‘red’, ‘green’, ‘blue’] is a list containing three string instances.
- The list( ) constructor produces an empty list by default.
- The list constructor will accept any iterable parameter.
  - list(‘hello’) produces a list of individual characters, [‘h’, ‘e’, ‘l’, ‘l’, ‘o’].

The tuple Class

- The **tuple** class provides an immutable (unchangeable) version of a sequence, which allows instances to have an internal representation that may be more streamlined than that of a list. Parentheses delimit a tuple.
  - The empty tuple is ()
- To express a tuple of length one as a literal, a comma must be placed after the element, but within the parentheses.
  - For example, (17,) is a one-element tuple.

The str Class

- String literals can be enclosed in single quotes, as in ‘hello’, or double quotes, as in "hello".
- A string can also begin and end with three single or double quotes, if it contains newlines in it.

```
print("""Welcome to the GPA calculator.
Please enter all your letter grades, one per line.
Enter a blank line to designate the end.""")
```

The set Class

- Python's **set** class represents a set, namely a collection of elements, without duplicates, and without an inherent order to those elements.
- Only instances of immutable types can be added to a Python set. Therefore, objects such as integers, floating-point numbers, and character strings are eligible to be elements of a set.
  - The frozenset class is an immutable form of the set type, itself.
- Python uses curly braces { and } as delimiters for a set
  - For example, as {17} or {'red', 'green', 'blue'}
  - The exception to this rule is that { } does not represent an empty set. Instead, the constructor set( ) returns an empty set.

The dict Class

- Python's dict class represents a dictionary, or mapping, from a set of distinct keys to associated values.
- Python implements a dict using an almost identical approach to that of a set, but with storage of the associated values.
  - The literal form { } produces an empty dictionary.
- A nonempty dictionary is expressed using a comma-separated series of key:value pairs. For example, the dictionary {'ga' : 'Irish', 'de' : 'German'} maps 'ga' to 'Irish' and 'de' to 'German'.
- Alternatively, the constructor accepts a sequence of key-value pairs as a parameter, as in dict(pairs) with pairs = [('ga', 'Irish'), ('de', 'German')].

Expressions and Operators

- Existing values can be combined into expressions using special symbols and keywords known as operators.
- The semantics of an operator depends upon the type of its operands.
  - For example, when a and b are numbers, the syntax a + b indicates addition,
  - while if a and b are strings, the operator + indicates concatenation.

Logical Operators

- Python supports the following keyword operators for Boolean values:

not	unary negation
and	conditional and
or	conditional or

- The and and or operators short-circuit, in that they do not evaluate the second operand if the result can be determined based on the value of the first operand.

Equality Operators

- Python supports the following operators to test two notions of equality:

is	same identity
is not	different identity
==	equivalent
!=	not equivalent

- The expression, a is b, evaluates to True, precisely when identifiers a and b are aliases for the same object.
- The expression a == b tests a more general notion of equivalence.

Comparison Operators

- Data types may define a natural order via the following operators:

$<$	less than
$<=$	less than or equal to
$>$	greater than
$>=$	greater than or equal to

- These operators have expected behavior for numeric types, and are defined lexicographically, and case-sensitively, for strings.

Arithmetic Operators

- Python supports the following arithmetic operators:

+	addition
−	subtraction
*	multiplication
/	true division
//	integer division
%	the modulo operator

- For addition, subtraction, and multiplication, if both operands have type int, then the result is an int; if one or both operands have type float, the result is a float.
- True division is always of type float, integer division is always int (with the result truncated)

Bitwise Operators

- Python provides the following bitwise operators for integers:

$\sim$	bitwise complement (prefix unary operator)
&	bitwise and
|	bitwise or
^	bitwise exclusive-or
<<	shift bits left, filling in with zeros
>>	shift bits right, filling in with sign bit

Sequence Operators

- Each of Python's built-in sequence types (str, tuple, and list) support the following operator syntaxes:

s[j]	element at index *j*
s[start:stop]	slice including indices [start,stop)
s[start:stop:step]	slice including indices start, start + step, start + 2*step, . . . , up to but not equalling or stop
s + t	concatenation of sequences
k * s	shorthand for s + s + s + ... (k times)
val **in** s	containment check
val **not in** s	non-containment check

Sequence Comparisons

- Sequences define comparison operations based on lexicographic order, performing an element by element comparison until the first difference is found.
  - For example, **[5, 6, 9]** < **[5, 7]** because of the entries at index 1.

`s == t`	equivalent (element by element)
`s != t`	not equivalent
`s < t`	lexicographically less than
`s <= t`	lexicographically less than or equal to
`s > t`	lexicographically greater than
`s >= t`	lexicographically greater than or equal to

Operators for Sets

- Sets and frozensets support the following operators:

key **in** s	containment check
key **not in** s	non-containment check
s1 == s2	s1 is equivalent to s2
s1 != s2	s1 is not equivalent to s2
s1 <= s2	s1 is subset of s2
s1 < s2	s1 is proper subset of s2
s1 >= s2	s1 is superset of s2
s1 > s2	s1 is proper superset of s2
s1 | s2	the union of s1 and s2
s1 & s2	the intersection of s1 and s2
s1 − s2	the set of elements in s1 but not s2
s1 ^ s2	the set of elements in precisely one of s1 or s2

Operators for Dictionaries

- The supported operators for objects of type dict are as follows:

d[key]	value associated with given key
d[key] = value	set (or reset) the value associated with given key
del d[key]	remove key and its associated value from dictionary
key **in** d	containment check
key **not in** d	non-containment check
d1 == d2	d1 is equivalent to d2
d1 != d2	d1 is not equivalent to d2

Operator Precedence

Operator Precedence		
	Type	**Symbols**
1	member access	expr.member
2	function/method calls container subscripts/slices	expr(...) expr[...]
3	exponentiation	**
4	unary operators	+expr, −expr, ~expr
5	multiplication, division	*, /, //, %
6	addition, subtraction	+, −
7	bitwise shifting	<<, >>
8	bitwise-and	&
9	bitwise-xor	^
10	bitwise-or	|
11	comparisons containment	**is**, **is not**, ==, !=, <, <=, >, >= **in**, **not in**
12	logical-not	**not** expr
13	logical-and	**and**
14	logical-or	**or**
15	conditional	val1 **if** cond **else** val2
16	assignments	=, +=, −=, *=, etc.