1/19/22 Notes

Tuesday, January 25, 2022 2:27 PM

Python Primer 3: Control Flow

Last class

- sequences: str, list, tuple
  - mutable [1, 1, 2, 2, 3], (1, 1, 2, 2, 3)
  - elements can repeat
  - can be a negative index
    - negative starts with the length of the sequence
    - length = 5
    - negative starts with -5
- collection: sets, frozensets
  - immutable {4.0, 3.3, 3}
  - no repeats
  - no order
  - no index
- key to value mapping: dictionary
  - mutable {'Jack':4.0, 'Jo':3.3, 'Jill':3.8}
  - key = name
  - value = number

Sequence Operators

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

		s = "FAU"
		t = "OWLS"

s[j]	element at index j	s[1] t[1] "A" "S"
s[start:stop]	slice including indices [start,stop) - excludes anything after	t[0:3] OWL
s[start:stop:step]	slice including indices start, start + step, start + 2*step, … up to but not equaling stop - larger string --> steps repeat	s[0:3:2] "FU"
s + t	concatenation of sequence - must be the same type	"FAUOWLS"
k * s	shorthand for s + s + s + … k times	k = 3 "FAUFAUFAU"
val in s	contaminent check - find value in sequence	
val not in s	non-contaminent check	

- convert to tuple
  - tuple(s)
- convert to list
  - list('!')

Sequence Comparisons

- sequences define comparison operations based on lexicographic (alphabetical) order
- perform an element by element comparison until the first differnce is found
  - [5, 6, **9**] < [5, 7]
    - even though the left is longer, 6 < 7
    - smaller value, not length

		s = "FAU"
		t = "OWLS"

s == t	equivalent (element by element)	False
s != t	not equivalent	True
s < t	lexicographically less than	True, F < O
s <= t	lexicographically less than or equal to	True
s > t	lexicographically greater than	False
s > = t	lexicographically greater than or equal to	False

Operators for sets

- non-repeating
- sets & frozen sets support the following operators
- set(" … ")- constructor

			s1 = set("FAU OWLS)
			s2 = set("GO OWLS")

	key in s	contaminent check	
	key not in s	non-contaminent check	
	s1 == s 2	s1 is equivalent to s2	False
	s1 != s2	s1 is not equivalent to s2	
	s1 <= s2	**s1 is a subset of s2**	
	s1 < s2	**s1 is a proper subset of s2**	
	s1 >= s2	**s1 is a superset of s2**	
bool ↑	s1 > s2	**s1 is a proper superset of s2**	
set element ↓	s1 | s2	the **union** of s1 and s2 - gets rid of duplicates - print unique elements - random order	{'A', 'L', 'O', 'W', 'G', 'U', 'U', 'F'}
	s1 & s2	the **intersection** of s1 and s2 - values in both	{'L', 'S', 'W', 'O', ' '}
	s1 - s2	the set of elements **in s1 but not s2 differentiation** - unique in s1	{'A', 'U', 'F'}
	s1 ^ s2	the set of elements in either s1 or s2 not both - removes elements that are in both	{'A', 'U','F', 'G'}

Operators for dictionary

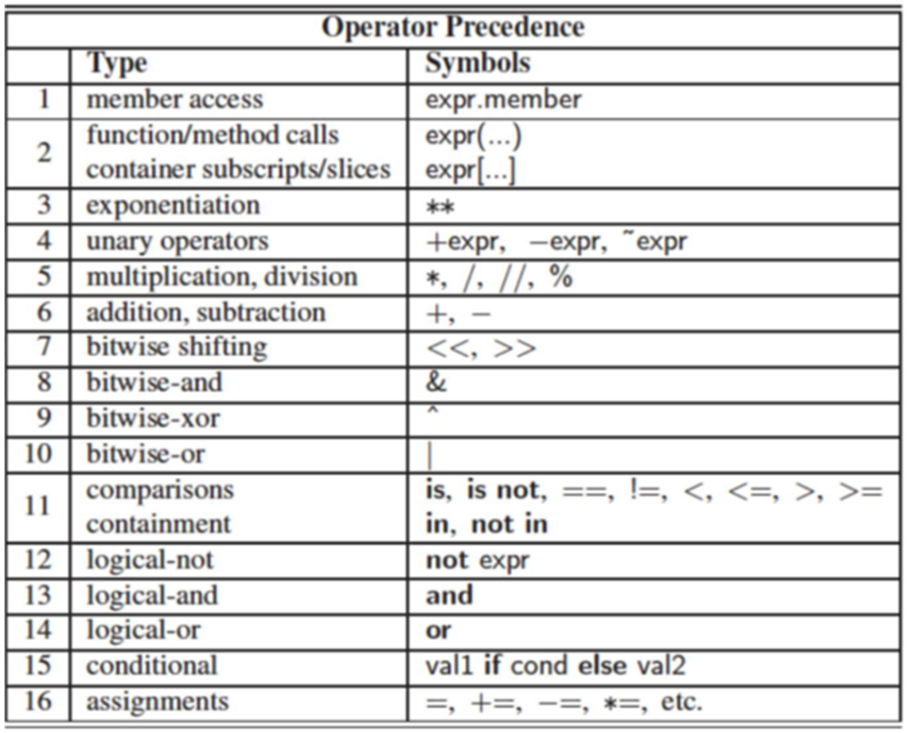
- the supported operators for objects of type dict

		{'Jack':'A+', 'Jo':'B-', 'Jill':'B'}
d[key]	value associated with given key	d['Jack'] = 'A+'
d[key] = value	set/reset the value associated with given key - changes value	d['Jack'] = 'F' d = {'Jack':'F', 'Jo':'B', 'Jill':'B-'}
del d[key]	remove key and its associated value from dictionary	del d['Jack']
k in d	contaminent check	'Jake' in d False
d1 == d2	d1 is equivalent to d2 - compare	
d1 != d2	d1 is not equivalent to d2 - compare	

- powerful & valuable keys
- keys are unique
  - values can repeat
  - 2 keys can point to same address
- keys can be any type
- set- throws out duplicates
  - use for a key

Chained Assignment

- x = y= 0
  - both pointing to 0
  - assigns multiple identifiers to the rightmost value
- chaining of comparison operators
  - 1 <= x + y <= 10
    - can do in Python, not C
  - equivalent to 1 <= x + y and x + y <= 10



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.

Simple Output

- print
  - built-in function
  - used to generate standard output to the console
  - simplest form
    - prints an arbitrary sequence of arguments
    - separated by spaces
    - followed by a trailing new line character
    - **print('maroon', 5)**
    - outputs the string: 'marron 5\n'
      - automatic space
  - a non string argument x will be displayed as **str(x)**
  - without any arguments, the command print( ) outputs a single new line character
  - by default items are printed with space in between them
  - keyword **sep** to separate outputs
    - **print(a, b, c, sep= ':')**
    - colon separated print list
  - keyword **end** to change the \n
    - **print(a, b, c, end= ' ')**

Simple Input

- primary means for acquiring info from the user console
  - function **input**
    - displays a prompt, even if given as an optional parameter
    - then waits until the user enters some sequence of characters followed by the return key
    - the return value of the function is the **string of characters** that were entered strictly before the return key
      - such a string can immediately be converted
      - **year = int(input('In what year were you born? '))**