*Natural Language Procesing with Python Chapter 4*

Accessing Text

|  |  |
| --- | --- |
| Problem | Solution |
| List variables point to references (not values) | listSentence = "I cant stand public servants".split(' ')  duplicateSentence = listSentence  listSentence[4] = "employees"  duplicateSentence >>> ['I', 'cant', 'stand', 'public', 'employees'] |
| Using the id() operator [*from above]* | id(duplicateSentence) >>> 4336120392  id(duplicateSentence) == id(listSentence) >>> true |
| Using “is” | duplicateSentence is listSentence >>> true  mark = list(['I', 'cant', 'stand', 'public', 'employees'])  mark is duplicateSentence >>> false  mark == duplicateSentence >>> true |
| Testing for undefined elements | testList = ['','cat', 'dog','']  for item in testList:  if item:  print("hey is an item " + item)  hey is an item cat  hey is an item dog |
| elif semantics | Python will never try to evaluate the elif if the if statement is true. Elif only works if the preceding if statement is false and the elif is true. |
| all() and any() | all(len(w) > 4 for w in lstTest)  any(len(w) > 4 for w in lstTest) |
| tuples (defined with commas) | tup = 'mark', 'big', 3, 'koala'  tup >> ('mark', 'big', 3, 'koala') |
| Various sequences | for item in sorted(s)  for item in set(s)  for item in reverse(s)  for item in set(s).difference(t)  for item in random.shuffle(s) |
| Multiple Assignment (using tuples) | words[2], words[1], words[4] = words[3], words[4], words[10] |
| Using zip (merging lists into tuples) | list1 = ['one','two','three']  list2 = ['ichi', 'ni', 'san']  zip(list1,list2) >>> [('one', 'ichi'), ('two', 'ni'), ('three', 'san')] |
| Enumerate (assigns ordinal numbers to a list) | list(enumerate(list1)) >>> [(0, 'one'), (1, 'two'), (2, 'three')] |
| Cut up a list into two (partitions) | testList = "one two three four five six seven".split(' ')  cut = 3  list1, list2 = testList[:cut], testList[cut:]  list1, list2 >> (['one', 'two', 'three'], ['four', 'five', 'six', 'seven']) |
| Use of blank tuple (uses \_ [i.e. underscore by convention] | words = "mark the koala climbs up trees".split()  wordlengths = [(len(word),word) for word in words]  wordlengths.sort()  ' '.join(w for (\_,w) in wordlengths) >>> 'up the mark koala trees climbs' |
| Generator expressions Sometimes quicker as the second sytax streams the results into the max funciton while the first syntax allocates memory first | max([w.lower() for w in nltk.word\_tokenize(text)]) #uses comprehension  max(w.lower() for w in nltk.word\_tokenize(text)) #uses generator |
| Procedural Versus Declarative | Procedural uses for loops while declarative users comprehensions and other higher level items. Program efficiency versus programmer efficiency. |

|  |  |
| --- | --- |
| Python and iteration | It is tempting to adopt idioms from other languages but Python offers some more elegant alternatives. |
| Docstrings for functions (on the first line) | def circle(radius):  """this is a function to calculate a circle"""  return(radius ^ 2 \* 3.14)  help(circle)  circle(radius)  this is a function to calculate a circle |
| Single element list | mark = ['hatcher']  mark.append('mark')  mark  ['hatcher', 'mark'] |
| Passing by value or reference | Lists and other sequences are passed by reference, while, scalars are passed by value. This means that lists and be changed within the function. |
| A function can create a global variable. | Using the **global** declaration. |
| Python basestring | This generalises to string and unicode types |
| Assert statement | The assert statement if it fails will produce and error and hald execution |
| Assertion (example) | def tag(word):  assert isinstance(word,basestring), "argument must be a string"  print("well we passed the assertion")  tag("mark")  well we passed the assertion  tag(14)  *Many error messages* |
| Functions | Allow us to group multiple actions into a single complex action and associate a name with it. |
| Documentation example | Page 148 |
|  |  |
|  |  |
|  |  |
|  |  |