Data types in ML using Python

**Data types :**

|  |
| --- |
| 1. **Lists** |
| 1. **Tuples** |
| 1. **Strings** |
| 1. **Dictionaries** |
| 1. **Sets** |

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **# determine the type of an object**  type(2) # returns 'int'  type(2.0) # returns 'float'  type('two') # returns 'str'  type(True) # returns 'bool'  type(None) # returns 'NoneType'  **# check if an object is of a given type**  isinstance(2.0, int) # returns False  isinstance(2.0, (int, float)) # returns True  **# convert an object to a given type**  float(2)  int(2.9)  str(2.9)  **# zero, None, and empty containers are converted to** False  bool(0)  bool(None)  bool('') # empty string  bool([]) # empty list  bool({}) # empty dictionary  **# non-empty containers and non-zeros are converted** to True  bool(2)  bool('two')  bool([2]) | |  |

**1) Lists :**

Different objects categorized along a certain ordered sequence, lists are ordered, iterable, mutable

(adding or removing objects changes the list size), can contain multiple data types.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72 | **# create an empty list (two ways)**  empty\_list = []  empty\_list = list()  **# create a list**  simpsons = ['homer', 'marge', 'bart']  **# examine a list**  simpsons[0] # print element 0 ('homer')  len(simpsons) # returns the length (3)  **# modify a list (does not return the list)**  simpsons.append('lisa') # append element to end  simpsons.extend(['itchy', 'scratchy']) # append multiple elements to end  simpsons.insert(0, 'maggie') # insert element at index 0 (shifts everything␣  ˓→right)  simpsons.remove('bart') # searches for first instance and removes it  simpsons.pop(0) # removes element 0 and returns it  del simpsons[0] # removes element 0 (does not return it)  simpsons[0] = 'krusty' # replace element 0  **# concatenate lists (slower than 'extend' method)**  neighbors = simpsons + ['ned','rod','todd']  **# find elements in a list**  simpsons.count('lisa') # counts the number of instances  simpsons.index('itchy') # returns index of first instance  **# list slicing [start:end:stride]**  weekdays = ['mon','tues','wed','thurs','fri']  weekdays[0] # element 0  weekdays[0:3] # elements 0, 1, 2  weekdays[:3] # elements 0, 1, 2  weekdays[3:] # elements 3, 4  weekdays[-1] # last element (element 4)  weekdays[::2] # every 2nd element (0, 2, 4)  weekdays[::-1] # backwards (4, 3, 2, 1, 0)  **# alternative method for returning the list backwards**  list(reversed(weekdays))  **# sort a list in place (modifies but does not return the list)**  simpsons.sort()  simpsons.sort(reverse=True) # sort in reverse  simpsons.sort(key=len) # sort by a key  # return a sorted list (but does not modify the original list)  sorted(simpsons)  sorted(simpsons, reverse=True)  sorted(simpsons, key=len)  **# create a second reference to the same list**  num = [1, 2, 3]  same\_num = num  same\_num[0] = 0 # modifies both 'num' and 'same\_num'  **# copy a list (three ways)**  new\_num = num.copy()  new\_num = num[:]  new\_num = list(num)  **# examine objects**  id(num) == id(same\_num) # returns True  id(num) == id(new\_num) # returns False  num is same\_num # returns True  num is new\_num # returns False  num == same\_num # returns True  num == new\_num # returns True (their contents are equivalent)  **# conatenate +, replicate \***  [1, 2, 3] + [4, 5, 6]  ["a"] \* 2 + ["b"] \* 3 |

**2) Tuples :**

Like lists, but their size cannot change: ordered, iterable, immutable, can contain multiple data

Types

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | **#Tuples**  **# create a tuple**  digits = (0, 1, 'two') # create a tuple directly  digits = tuple([0, 1, 'two']) # create a tuple from a list  zero = (0,) # trailing comma is required to indicate it's a tuple  **# examine a tuple**  digits[2] # returns 'two'  len(digits) # returns 3  digits.count(0) # counts the number of instances of that value (1)  digits.index(1) # returns the index of the first instance of that value (1)  **# elements of a tuple cannot be modified**  **# digits[2] = 2 # throws an error**  **# concatenate tuples**  digits = digits + (3, 4)  **# create a single tuple with elements repeated (also works with lists)**  (3, 4) \* 2 # returns (3, 4, 3, 4)  **# tuple unpacking**  bart = ('male', 10, 'simpson') # create a tuple | |

**3) Strings :**

A sequence of characters, they are iterable, immutable

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | **## Strings**  **# create a string**  s = str(42) # convert another data type into a string  s = 'I like you'  **# examine a string**  s[0] # returns 'I'  len(s) # returns 10  # string slicing like lists  s[:6] # returns 'I like'  s[7:] # returns 'you'  s[-1] # returns 'u'  **# basic string methods (does not modify the original string)**  s.lower() # returns 'i like you'  s.upper() # returns 'I LIKE YOU'  s.startswith('I') # returns True  s.endswith('you') # returns True  s.isdigit() # returns False (returns True if every character in the string is a␣  ˓→digit)  s.find('like') # returns index of first occurrence (2), but doesn't support regex  s.find('hate') # returns -1 since not found  s.replace('like','love') # replaces all instances of 'like' with 'love'  **# split a string into a list of substrings separated by a delimiter**  s.split(' ') # returns ['I','like','you']  s.split() # same thing  s2 = 'a, an, the'  s2.split(',') # returns ['a',' an',' the']  **# join a list of strings into one string using a delimiter**  stooges = ['larry','curly','moe']  ' '.join(stooges) # returns 'larry curly moe'  **# concatenate strings**  s3 = 'The meaning of life is'  s4 = '42'  s3 + ' ' + s4 # returns 'The meaning of life is 42'  s3 + ' ' + str(42) # same thing  **# remove whitespace from start and end of a string**  s5 = ' ham and cheese '  s5.strip() # returns 'ham and cheese'  **# string substitutions: all of these return 'raining cats and dogs'**  'raining %s and %s' % ('cats','dogs') # old way  'raining {} and {}'.format('cats','dogs') # new way  'raining {arg1} and {arg2}'.format(arg1='cats',arg2='dogs') # named arguments |

**4) Dictionaries :**

Dictionaries are structures which can contain multiple data types, and is ordered with key-value

pairs: for each (unique) key, the dictionary outputs one value. Keys can be strings, numbers, or

tuples, while the corresponding values can be any Python object. Dictionaries are: unordered,

iterable, mutable.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | **##Dictionaries**  **# create an empty dictionary (two ways)**  empty\_dict = {}  empty\_dict = dict()  **# create a dictionary (two ways)**  family = {'dad':'homer', 'mom':'marge', 'size':6}  family = dict(dad='homer', mom='marge', size=6)  **# convert a list of tuples into a dictionary**  list\_of\_tuples = [('dad','homer'), ('mom','marge'), ('size', 6)]  family = dict(list\_of\_tuples)  **# examine a dictionary**  family['dad'] # returns 'homer'  len(family) # returns 3  family.keys() # returns list: ['dad', 'mom', 'size']  family.values() # returns list: ['homer', 'marge', 6]  family.items() # returns list of tuples:  **# [('dad', 'homer'), ('mom', 'marge'), ('size', 6)]**  'mom' in family # returns True  'marge' in family # returns False (only checks keys)  **# modify a dictionary (does not return the dictionary)**  family['cat'] = 'snowball' # add a new entry  family['cat'] = 'snowball ii' # edit an existing entry  del family['cat'] # delete an entry  family['kids'] = ['bart', 'lisa'] # value can be a list  family.pop('dad') # removes an entry and returns the value ('homer')  family.update({'baby':'maggie', 'grandpa':'abe'}) # add multiple entries |

**5) Sets :**

Like dictionaries, but with unique keys only (no corresponding values). They are: unordered, iterable,

mutable, can contain multiple data types made up of unique elements (strings, numbers, or tuples)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | **## Set**  **# create an empty set**  empty\_set = set()  **# create a set**  languages = {'python', 'r', 'java'} # create a set directly  snakes = set(['cobra', 'viper', 'python']) # create a set from a list  **# examine a set**  len(languages) # returns 3  'python' in languages # returns True  **# set operations**  languages & snakes # returns intersection: {'python'}  languages | snakes # returns union: {'cobra', 'r', 'java', 'viper', 'python'}  languages - snakes # returns set difference: {'r', 'java'}  snakes - languages # returns set difference: {'cobra', 'viper'}  **# modify a set (does not return the set)**  languages.add('sql') # add a new element  languages.add('r') # try to add an existing element (ignored, no error)  languages.remove('java') # remove an element  try:  languages.remove('c') # try to remove a non-existing element (throws an error)  except KeyError as e:  print("Error", e)  languages.discard('c') # removes an element if present, but ignored otherwise  languages.pop() # removes and returns an arbitrary element  languages.clear() # removes all elements  languages.update('go', 'spark') # add multiple elements (can also pass a list or set)  **# get a sorted list of unique elements from a list**  sorted(set([9, 0, 2, 1, 0])) # returns [0, 1, 2, 9] |