**PYTHON DATA SCIENCE TOOLBOX**

**PART-1**

|  |
| --- |
| Unpacking tuples |
| In [1]: even\_nums = (2, 4, 6)  In [2]: a, b, c = even\_nums |
| Scope and user-defined functions |
| # Scope - part of the program where an object or name may be  accessible  ● Global scope - defined in the main body of a script  ● Local scope - defined inside a function  ● Built-in scope - names in the pre-defined built-ins  module |
| global |
| In [1]: new\_val = 10  In [2]: def square(value):  ...: """Returns the square of a number."""  ...: global new\_val  ...: new\_val = new\_val \*\* 2  ...: return new\_val |
| nonlocal |
| def outer():  """Prints the value of n."""  n = 1  def inner():  nonlocal n  n = 2  print(n)  inner()  print(n) |
| Nested functions |
| def raise\_val(n):  """Return the inner function."""  def inner(x):  """Raise x to the power of n."""  raised = x \*\* n  return raised  return inner  In [1]: square = raise\_val(2)  In [2]: cube = raise\_val(3)  In [3]: print(square(2), cube(4))  4 64 |
| Scopes searched  ● Local scope  ● Enclosing functions  ● Global  ● Built-in |

|  |
| --- |
| Default and flexible arguments |
| Add a default argument |
| In [1]: def power(number, pow=1):  ....: """Raise number to the power of pow."""  ....: new\_value = number \*\* pow  ....: return new\_value  In [2]: power(9, 2)  Out [2]: 81  In [3]: power(9, 1)  Out [3]: 9  In [4]: power(9)  Out[4]: 9 |
| \*args (1) |
| def add\_all(\*args):  """Sum all values in \*args together."""  # Initialize sum  sum\_all = 0  # Accumulate the sum  for num in args:  sum\_all += num  return sum\_all |
| \*\*kwargs |
| In [1]: print\_all(name="Hugo Bowne-Anderson", employer="DataCamp")  name: Hugo Bowne-Anderson  employer: DataCamp  def print\_all(\*\*kwargs):  """Print out key-value pairs in \*\*kwargs."""  # Print out the key-value pairs  For key, value in kwargs.items():  print(key + ": " + value)  In [1]: print\_all(name="dumbledore", job="headmaster")  job: headmaster  name: dumbledore |

|  |
| --- |
| Lambda |
| In [1]: raise\_to\_power = lambda x, y: x \*\* y  In [2]: raise\_to\_power(2, 3)  Out[2]: 8 |
| Map() and lambda functions |
| # Function map takes two arguments: map(func, seq)  # map() applies the function to ALL elements in the sequence |
| In [1]: nums = [48, 6, 9, 21, 1]  In [2]: square\_all = map(lambda num: num \*\* 2, nums)  In [3]: print(square\_all)  <map object at 0x103e065c0>  In [4]: print(list(square\_all))  [2304, 36, 81, 441, 1] |
| Filter() and lambda functions |
| fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', 'legolas', 'gimli', 'gandalf']  result = filter(lambda member: len(member)>6, fellowship)  result\_list = list(result)  print(result\_list) |
| Reduce() and lambda functions |
| # The reduce() function is useful for performing some computation on a list and, unlike map() and filter(), returns a single value as a result. |
| from functools import reduce  stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']  result = reduce(lambda item1, item2: item1+item2, stark)  print(result) |

|  |
| --- |
| try-except clause |
| In [1]: def sqrt(x):  ...: """Returns the square root of a number."""  ...: try:  ...: return x \*\* 0.5  ...: except:  ...: print('x must be an int or float') |
| In [3]: def sqrt(x):  ...: """Returns the square root of a number."""  ...: if x < 0:  ...: raise ValueError('x must be non-negative')  ...: try:  ...: return x \*\* 0.5  ...: except TypeError:  ...: print('x must be an int or float') |

**PART-2**

|  |
| --- |
| Iterators in PythonLand |
| for loop |
| iter() / next() |
| In [1]: word = 'Da'  In [2]: it = iter(word)  In [3]: next(it)  Out[3]: 'D'  In [4]: next(it)  Out[4]: 'a'  In [5]: next(it) |
| In [1]: word = 'Data'  In [2]: it = iter(word)  In [3]: print(\*it)  D a t a  In [4]: print(\*it) # no values to go through |
| In [1]: file = open('file.txt')  In [2]: it = iter(file) # “it” is an iterator, produces next value with next()  In [3]: print(next(it))  This is the first line.  In [4]: print(next(it))  This is the second line. |
| list() / sum() |
| # these are iterator as function arguments |
| values = range(10, 21)  print(values)  values\_list = list(values)  print(values\_list)  values\_sum = sum(values)  range(10, 21)  [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]  165 |
| enumerate() |
| In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']  In [2]: e = enumerate(avengers)  In [3]: print(type(e))  <class 'enumerate'>  In [4]: e\_list = list(e)  In [5]: print(e\_list)  [(0, 'hawkeye'), (1, 'iron man'), (2, 'thor'), (3, 'quicksilver')] |
| # enumerate() and unpack |
| In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']  In [2]: for index, value in enumerate(avengers):  ....: print(index, value)  0 hawkeye  1 iron man  2 thor  3 quicksilver  In [3]: for index, value in enumerate(avengers, start=10):  ....: print(index, value)  10 hakweye  11 iron man  12 thor  13 quicksilver |
| zip() |
| In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']  In [2]: names = ['barton', 'stark', 'odinson', 'maximoff']  In [3]: z = zip(avengers, names)  In [4]: print(type(z))  <class 'zip'>  In [5]: z\_list = list(z)  In [6]: print(z\_list)  [('hawkeye', 'barton'), ('iron man', 'stark'), ('thor',  'odinson'), ('quicksilver', 'maximoff')] |
| # zip() and unpack |
| In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']  In [2]: names = ['barton', 'stark', 'odinson', 'maximoff']  In [3]: for z1, z2 in zip(avengers, names):  ....: print(z1, z2)  hawkeye barton  iron man stark  thor odinson  quicksilver maximoff |
| # result1, result2 = zip(\*z1) |
| # Print zip with \* |
| In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']  In [2]: names = ['barton', 'stark', 'odinson', 'maximoff']  In [3]: z = zip(avengers, names)  In [4]: print(\*z)  ('hawkeye', 'barton') ('iron man', 'stark') ('thor', 'odinson')  ('quicksilver', 'maximoff') |
| Using iterators for big data |
| chunksize |
| In [1]: import pandas as pd  In [2]: result = []  In [3]: for chunk in pd.read\_csv('data.csv', chunksize=1000):  ...: result.append(sum(chunk['x']))  In [4]: total = sum(result)  In [5]: print(total)  4252532 |
| In [1]: import pandas as pd  In [2]: total = 0  In [3]: for chunk in pd.read\_csv('data.csv', chunksize=1000):  ...: total += sum(chunk['x'])  In [4]: print(total)  4252532 |
| ########## |
| counts\_dict = {}  for chunk in pd.read\_csv('tweets.csv', chunksize=10):  for entry in chunk['lang']:  if entry in counts\_dict.keys():  counts\_dict[entry] += 1  else:  counts\_dict[entry] = 1  print(counts\_dict)  {'et': 1, 'en': 97, 'und': 2} |

|  |
| --- |
| List comprehensions |
| # for loop olusturulan listenin, tek satirda olusturulmasi |
| Populate a list with a for loop |
| In [1]: nums = [12, 8, 21, 3, 16]  In [2]: new\_nums = []  In [3]: for num in nums:  ...: new\_nums.append(num + 1)  In [4]: print(new\_nums)  [13, 9, 22, 4, 17] |
| A list comprehension |
| In [1]: nums = [12, 8, 21, 3, 16]  In [2]: new\_nums = [num + 1 for num in nums]  In [3]: print(new\_nums)  [13, 9, 22, 4, 17] |
| List comprehension with range() |
| In [1]: result = [num for num in range(11)]  In [2]: print(result)  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] |
| Nested loops with “for loop” |
| In [1]: pairs\_1 = []  In [2]: for num1 in range(0, 2):  ...: for num2 in range(6, 8):  ...: pairs\_1.append(num1, num2)  In [3]: print(pairs\_1)  [(0, 6), (0, 7), (1, 6), (1, 7)] |
| Nested loops with “list comprehension” |
| In [1]: pairs\_2 = [(num1, num2) for num1 in range(0, 2) for num2  in range(6, 8)]  In [2]: print(pairs\_2)  [(0, 6), (0, 7), (1, 6), (1, 7)] |
| Conditionals in comprehensions |
| In [1]: [num \*\* 2 for num in range(10) if num % 2 == 0]  Out[1]: [0, 4, 16, 36, 64] |
| In [2]: [num \*\* 2 if num % 2 == 0 else 0 for num in range(10)]  Out[2]: [0, 0, 4, 0, 16, 0, 36, 0, 64, 0] |
| Dict comprehensions |
| In [1]: pos\_neg = {num: -num for num in range(9)}  In [2]: print(pos\_neg)  {0: 0, 1: -1, 2: -2, 3: -3, 4: -4, 5: -5, 6: -6, 7: -7, 8: -8}  In [3]: print(type(pos\_neg))  <class 'dict'> |
| generators |
| Generator expressions |
| # Recall list comprehension, use ( ) instead of [ ],  # Generators - returns a generator object |
| In [2]: (2 \* num for num in range(10))  Out[2]: <generator object <genexpr> at 0x1046bf888> |
| Printing values from generators |
| In [1]: result = (num for num in range(6))  In [2]: for num in result:  ....: print(num)  0  1  2  3  4  5 |
| In [1]: result = (num for num in range(6))  In [2]: print(list(result))  [0, 1, 2, 3, 4, 5] |
| Conditionals in generator expressions |
| In [1]: even\_nums = (num for num in range(10) if num % 2 == 0)  In [2]: print(list(even\_nums))  [0, 2, 4, 6, 8] |
| Build a generator function |
| def num\_sequence(n):  """Generate values from 0 to n."""  i = 0  while i < n:  yield i  i += 1 |
| In [1]: result = num\_sequence(5)  In [2]: print(type(result))  <class 'generator'>  In [3]: for item in result:  ....: print(item)  0  1  2  3  4 |
| ########### |
| fellowship = ['frodo', 'samwise', 'merry', 'aragorn', 'legolas', 'boromir', 'gimli']  new\_fellowship = [member if len(member)>=7 else '' for member in fellowship]  print(new\_fellowship)  ['', 'samwise', '', 'aragorn', 'legolas', 'boromir', ''] |