|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **­Values/data types:**  To get data type – print type  Output:  type <’str’>, list, float, int, bool, dict  **Strings:**  Strings are immutable  Concatenation: joins two operands by linking them end-to-end (+)  \* performs repetition  *Ex:* *‘Fun’\*3 = FunFunFun*  Repetition has precedence over concatenation  **Slice:**  Substring of a string  [n:m] m excludes the last number  **Operators and operands:**  \* = multiplication  \*\* = exponentiation  / = division operator, produces floating point if either of operands are float, if both are int, performs integer division & truncates result down to next smallest integer  *Ex:* 9/5 = 1 vs. 9.0/5 = 1.8  // = integer division  *Ex: 7.0//3.0 = 2.0*  % = modulus operator, gives remainder of division problem  *Ex: 7%3 = 1*  **Dictionaries:**  .keys() returns list of keys  .values() returns list of associated values  .items() returns list of key-value pairs  In/Not in functions check if a **key** is in a dictionary  del function removes a key-value pair from the dictionary  *Ex: del inventory[‘pears’] removes the key pears from the dictionary inventory*  **Accumulating the Max:**  To find the maximum value:  *d = some dictionary*  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%201.40.57%*a = 0*  *for c in d:*  *if d[c] > a:*  *a = d[c]*  Accumulating the best key associated with the max val:  *d = some dictionary*  *ks = d.keys()*  *best\_so\_far = ks[0]*  *for k in ks:*  *if d[k] > d[best\_so\_far]:*  *best\_so\_far = k*  **Tuples are NOT mutable.**  **Sorting Lists:**  Sorted(list)  With integers: will sort from smallest to largest  With strings: will sort alphabetical  To sort reverse of these:  Sorted (list, reverse=True)  **Lambda functions:**  Shorter way of assigning a function  Lambda x: x-2 is applying the function x-2 to every x  **File Methods:**  open(‘filename’, ‘r’) 🡪 *open a file called filename and use it for reading, returns a reference to a file object*  open(‘filename’, ‘w’) 🡪 *open a file called filename and use it for writing, returns a reference to a file object*  filevariable.close() 🡪 *file use is complete*  readlines() 🡪 *returns a list of lines of text to iterate through*  Recipe for reading a file:  fname = “yourfile.txt”  fileref = open(fname, “r”)  fileref.close()  **lstA = [106]**  **lstB = lst A**  **lstA.append(206)**  **lstA = lstA[1:]**  **lstA.append(506)**  **print lstB**  [106, 206] | **Git**:  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.31.21%  git clone is opposite of git init 🡪creating repository on website, then you want to connect to it remotely  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.31.45%  git add 🡪certain files, sep by space  git commit –m “comment”  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.32.49%  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.33.21%../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.33.31%../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.34.03% |  | **Try/Except:**  Try to execute a block of code, if the block of code executes without any run-time errors, carry on with the rest of the program after try/except. If a runtime error does occur, skip the rest of that block of code, execute a block of code in the “except” clause, then carry on with rest of program after try/except statement.  Optional error restrictions (all other errors will cause program to stop running):  except Exception: 🡪 all  except IndexError: 🡪 trying to access index out of range  except ZeroDivisionError: 🡪 if you try to divide by zero  except KeyError: 🡪 a key is missing from a dict  If you print a variable along with exception, error info will print  *except Exception, e:*  *print “got error”*  *print e*  output: got error  IndexError: list index out of range on line 3  **Cache**  CACHE\_FNAME = 'cache.json'  try:      cache\_file = open(CACHE\_FNAME, 'r')      cache\_contents = cache\_file.read()      cache\_file.close()      CACHE\_DICTION = json.loads(cache\_contents)  except:      CACHE\_DICTION = {}  **Processing JSON results:**  json.loads() takes a string as input and produces a python object (list or dict)  *import requests*  *import json*  *result = requests.get(“url”, params = {“format”: “json”})*  *d = json.loads(result.text)*  d becomes a nicely formatted dictionary of the information requested  json.dumps() takes a python object and returns a string in json format  **Making requests:**  Use the get function in the requests module to fetch the contents of a page  *import requests*  *page = requests.get(“url”)*  Once we run this we get a response object  most websites that accept URL parameters in this form will accept the key-value pairs in any order | **Python Regular Expression Quick Guide**    **^**  Matches the beginning of a line  **$**  Matches the end of the line  **.**  Matches any character  **\s** Matches whitespace  **\S** Matches any non-whitespace character  **\***  Repeats a character zero or more times  **\*?**  Repeats a character zero or more times (non-greedy)  **+**  Repeats a character one or more times  **+?**  Repeats a character one or more times(non-greedy)  **[aeiou]** Matches a single character in the listed set  **[^XYZ]** Matches a single character not in the listed set  **[a-z0-9]** The set of characters can include a range  **(**  Indicates where string extraction is to start  **)**  Indicates where string extraction is to end  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%201.38.12%  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%201.38.20%  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%201.38.50%  Project 2: find all urls which must start with http:// or https:// and have at least 2 characters after the .  **urls = re.findall('http[s]?://\S+\.\S{2,}', s)**  **Beautiful Soup**  *From bs4 import BeautifulSoup*  *read = requests.get(url)*  *soup = BeautifulSoup(read.content, "html.parser")*  Three Sisters Ex:  soup.find\_all(‘a’)  *[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,*  *<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,*  *<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]*  soup.find(id="link3")  *# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>*  Extracting URLs:  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.07.46%  Using .find for a specific class (i.e. most popular)  ../../../../../Desktop/Screen%20Shot%202017-11-05%20at%202.18.26% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Many people keep time using a 24 hour clock. If it is currently 13 and you set your alarm to go off in 50 hours, it will be 15 (3pm). Ask the user for the time now (in hours), and then ask for the number of hours to wait for the alarm. Your program should output what the time will be on the clock when the alarm goes off.**  time\_now = raw\_input("What time is it now?")  hours = raw\_input("How many hours do you want to wait for the alarm?")  alarm = (int(time\_now) + int(hours)) % 24  print alarm  **Write a code that will print the reversed version of the variable ‘word’.**  word = ‘hello106’  reverse = “ ”  For x in word:  reverse = x + reverse  Print reverse  *Remember: put x where you want to add it, so in the final loop you want 6 to be the first letter not the last, so it would be x + reverse not reverse + x*  **What will print when the following code runs?**  **c1 = (‘P’, 30439)**  **c2 = (‘A’, 92739)**  **c3 = (‘D’, 82728)**  **cs = [c1, c2, c3]**  **for item in cs:**  **x,y,z = item**  **print x**  **print y**  **print z**  Error because the code is trying to unpack a tuple of length 2 into 3 slots.  **What will print out when the following code is executed?**  **S = “2016-02-01T20:12:2”**  **Print S[S.find(“16”)]**  1 🡪 *find a string will return the index where the first character in the substring is found*  **s = “2016—02—01T20”**  **t = s.split(“— “)**  **print t**  [‘2016’, ‘02’, ‘01T20’]  *Removes the parameter*  **Given the dictionary medals that shows the medal counts for countries in Rio, sort them by medal count. Save the three countries with the highest medal count.**  countries = medals.keys()  alphabetical = sorted(countries, key=lamda c: medals[c])  top\_three = alphabetical[:2]  Macintosh HD:Users:avaweiner:Desktop:Screen Shot 2017-04-20 at 4.50.56 PM.png  **d = {‘a’: 1, 2: ‘b’, ‘c’: 3}**  **for k in [‘a’, ‘b’, ‘c’]:**  **try:**  **print d[k]**  **except:**  **print “oops”**  **print “done”**  1  oops  3  done | s = “ “ “  [  {  “id”: 672149962780749800, “user” : { “followers\_count”: 823, “screen\_name”: “alimahrer”, “id”: 156844181  }  },  {  “id”: 672149247811313700, “user”: { “followers\_count”: 1569, “screen\_name”: “CatchaJob\_us”, “id”: 2348426863  }  },  {  “id”: 672148783413731300, “user”: { “followers\_counts”: 15575, “screen\_name”: “GymCastic”, “id” : 812521788  }  }  ] “ “ “  What is the type of s?  **String**  What is the type of s.split(“user”)?  **List**  What will the following code print out?  print len(s.split(“user”))  **4**  What is the type of json.loads(s)?  **List**  **Write a function popular\_tweeters that will take a string like s and return a list of screen names in order based on their follower count.**  Def popular\_tweeters(s):  L = json.loads(s)  Sorted\_L = (L, key = lambda t: t[‘user’][‘followers\_count’], reverse = True)  Return [t[‘user’][‘screen\_name’] for t in sorted\_L]  s = “““ {  “numResults”: 3,  “tweets”: [  {  “text”: “Anyone else headed to NQ?”,  “user”: {  “followers\_count”: 489,  “screen\_name”: “samc\_99”  }  },  {“text”: “it’s snowing!”,  “user”: {  “followers\_count”: 1204,  “screen\_name”: “jackiec”  }  },  {  “text”: “excited for winter break!”,  “user”: {  “followers\_count”: 821,  “screen\_name”: “tsuyoshi\_13”  }  }  ]  }”””  What is the type of s?  String  What is the type of s[20]?  String  What is the type of s[‘numResults’]?  No type  What is the type of json.loads(s)?  Dictionary  **Create a class to represent an individual tweet** **in string s**  class Tweet():  def \_\_init\_\_(self, tweet\_dict):  self.followers=tweet\_dict[‘user’][‘followers\_count’]  self.handle=tweet\_dict[‘user’][‘screen\_name’]  self.status=tweet\_dict[‘text’]  def hasMorethan500Followers(self):  return self.followers>500  tweet\_data=json.loads(s)  tweet\_instances= []  for tweet\_d in tweet\_data[‘tweets’]  tweet\_instances.apped(Tweet(tweet\_d))  print tweet\_instances[0].status  **#prints** “Anyone else headed to NQ?”  print tweet\_instances[1].handle  **#prints** “jackiec”  print tweet\_instances[2].followers  **#prints** 821  print tweet\_instances[2].hasMoreThank500Followers()  **#prints** True  **Write a function that will take a list of tweet instances and return a list of the users in order based on their followers\_count**  def popular\_tweeters(tweets):  return sorted(tweets, key = lambda t: t.followers, reverse = True) | *Def interp(L, i):*  *Templ = “%s is at idx %d in a list with %d items”*  *Vals = (L[i], i, len(L))*  *Return templ % vals*  *Print interp([‘you, ‘are’, ‘a’, ‘genius’], 3)*  Genius is at idx 3 in a list with 4 items  **Caching format:**  import requests  import json  import pickle  cache\_fname = "cached\_results.txt"  try:  fobj = open(cache\_fname, 'r')  saved\_cache = pickle.load(fobj)  fobj.close()  except:  saved\_cache = {}  def canonical\_order(d):  alphabetized\_keys = sorted(d.keys())  res = []  for k in alphabetized\_keys:  res.append((k, d[k]))  return res  def requestURL(baseurl, params = {}):  req = requests.Request(method = 'GET', url = baseurl, params = canonical\_order(params))  prepped = req.prepare()  return prepped.url  def get\_with\_caching(base\_url, params\_diction, cache\_diction, cache\_fname):  full\_url = requestURL(base\_url, params\_diction)  # step 1  if full\_url in cache\_diction:  # step 2  print "retrieving cached result for " + full\_url  return cache\_diction[full\_url]  else:  # step 3  response = requests.get(base\_url, params=params\_diction)  print "adding cached result for " + full\_url  # add to the cache and save it permanently  cache\_diction[full\_url] = response.text  fobj = open(cache\_fname, "w")  pickle.dump(cache\_diction, fobj)  fobj.close()  return response.text  dest\_url = 'http://services.faa.gov/airport/status/DTW'  d = {'format': 'json'}  result\_text = get\_with\_caching(dest\_url, d, saved\_cache, cache\_fname)  print json.loads(result\_text)  def enum(L):  return zip(range(len(L)), L)  print enum([‘a’, ‘b’, ‘c’])  *[(0, ‘a’), (1, ‘b’), (2, ‘c’)]*  **Write code using the request module to retrieve this URL: http://runestone.org/API/get\_score? assignment\_id=24&student\_id=374**  requests.get(“http://runestone.org/API/get\_score”,  params = {‘student\_id’ :374, ‘assignment\_id’ :24})  **What will the following code print?**  **x = -1**  **y = -2**  **z = -3**  **def h(x, y = 2, z = 3):**  **print x, y, z**  **h(x=y, z=5)**  -2, 2, 5  What is the output of the following code?  def crypt(s, n=2, L = ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’):  rv = ‘’  for letter in s:  try:  idx = L.index(letter)  rotated\_idx = (idx+n)%len(L)  rv += L[rotated\_idx]  except:  rv += ‘?’  return rv  print crypt(‘CaT’)  **E?V**  print crypt(‘Z’, n=3)  **C** | **Define a function sorted\_by\_keys that takes a dict as input and returns a list of its values sorted in alphabetical order based on the keys of those values.**  def sorted\_by\_keys(d):  sorted\_keys = sorted(d.keys)  return [d[x] for x in sorted\_keys]  **Define a method for the Post class that will return a count of the number of people who both liked and commented on a post.**  Def enthusiast\_count(self):  a = self.likers()  b = self.commenters()  c = [x for x in a if x in b]  return len (c)  **Write code to create 2 instances of Post that have no likes or comments. The first instance should have the message “I love python”, the second should have “Gadzooks, I am a programmer now!”**  d1 = {“message”: “I love python”}  d2 = {“message”: “Gadzooks, I am a programmer now!”}  a = Post(d1)  b = Post(d2)  Fill in the blank box below to define canFly.  class Pet():  is\_pet = True  def\_\_init\_\_(self, name, pet\_type):  self.name = name  self.pet\_type = pet\_type  def canFly(self):  #Should return True if self’s pet\_type instance  #variable is ‘bird’ or ‘fly’ and False otherwise  return self.pet\_type == ‘bird’ or self.pet\_type== ‘fly’  class Cat(Pet):  is\_cat = True  def \_\_init\_\_(self, name, has\_claws):  Pet.\_\_init\_\_(self, name, ‘cat’)  self.clawed = has\_claws  def canScratch(self):  return self.clawed  garfield = Cat(‘Garfield’, False)  cheshire = Pet(‘Cheshire’, ‘cat’)  Write the value of the expressions. If there is an error when evaluating the expression’s value, write “error”.  Expression  Value  garfield.is\_cat  True  cheshire.is\_cat  (error)  cheshire.canScratch()  (error)  garfield.canScratch()  False  garfield.pet\_type  ‘cat’  cheshire.is\_pet  True  **Define a function join\_strings that takes two inputs, a list of strings and a separator string, and returns a new string with all of the strings added together separated by the sep string.**  def join\_strings(L, sep):  res = “ ”  for x in L:  res = res + x + sep  return res[:-1]  **Sorting a list of instances based on a method:**  animals = sorted(animals, key = lambda x: x.hunger)  x = -1  y = -2  z = -3  def h(x, y=2, z=3)  print x, y, z  **h(z = 5)**  *Error because x no parameter for x and it is required* |