|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **­Values/data types:**  String – characters in quotes  *Ex: “Hello, World!”*  Float – #’s w/ decimal point  *Ex: 4.3*  Int – integer, whole number  *Ex: 7*  To get data type – print type  **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*  **Slice String:**  Substring of string  *Ex:* x = “hey, hi, hello”, print x[0:3] 🡪 returns hey  **Function calls:**  Def square(x)  Return x\*x – squares #  Sub(x, y) – x minus y  **Statements/expressions:**  Len – returns # of characters in string  *Ex: len(s)*  **Input:**  Raw\_input – opens up prompt window  **Boolean:**  True/false  *Ex: 5 == 5 is True, 5==6 is False*  And – true if both = true  Or – true if either conditions are true  Not – negates Boolean expression  **Indexing:**  Selects character from string  *Ex: x = “hello” m = x[3] 🡪 returns position 3, letter L*  First position 0  Last position -1 or number in sequence  **In/Not in:**  In – tests if 1 string is substring of another, returns true/false  Not in – tests if not in a string, returns true/false  **Lists:**  Elements in []  Len(list) returns # of items in list  Len(list[x]) returns # of characters of an element of list  **Concatenation/Repetition:**  Adds lists together  Can do list \* x 🡪 prints list x times  **List slices:**  Same as slice operator for lists  *Ex:* list[x:y] 🡪 returns list from x to y  **Entropy:** how unpredictable an event outcome is- reaches max. when we don’t know what outcome is  **Surprisal:** property of event outcome that is unlikely to have occurred  **Adding to dictionary:**  Mydict[“new thing”] = x  X would be value  **Deleting in dictionary:**  Del mydict[‘key’]  function only\_with\_d that takes a list of dicts as input returns a potentially shorter list containing only those dicts that have value associated with key 'd'. Thus, for example, we should get the output in bold with the code below. Make sure that your function works for any L,not just the one given.  def only\_with\_d(list\_of\_ds):  acc = []  for d in list\_of\_ds:    if 'd' in d.keys():  acc.append(d)  return acc  L = [{'a':1, 'b':2, 'd':11},     {'a':4, 'b':5, 'd':12},     {'a':7, 'b':8, 'f':11}    ]  print only\_with\_d(L)  **[{'a':1, 'b':2, 'd':11}, {'a':4, 'b':5, 'd':12}]--------------------**  After following Python code is executed, what is type of n[:3]?  N = “12345”  **String**  After following Python code is executed, what is type of L[2:3]?  L = [‘h’, ‘e’, ‘l’, ‘l’, ‘o’]  **List**  What will this code print?  Myvar = “hello”  Print “myvar”[0]  **M**  After this is executed, what will be type of variable s?  S = “<published>2009-01-23T20:04:53Z</published>”  T = s.split()  **String & list**  After this is executed, what will be type of variable t?  S = “<published>2009-01-23T20:04:53Z</published>”  T = s.find(“2009”)  **Integer**  What will this print?  S = “<published>2009-01-23T20:04:53Z</published>”  Print s.split(‘-‘)[1]  **01**  What will this print?  S = “<published>2009-01-23T20:04:53Z</published>”  Print len(s.split(‘T’)[0].split(‘2’))  **3**  Consider 2 files. First has 3 verses, 1 chorus repeated after each verse. Second has 6 verses, all different from each other. Both files have same # of characters . Which file has more info, chorus or no chorus?  **File with no chorus has more info b/c more none repeated info.**  Consider 2 coins, one “fair”, lands on heads/tails 50/50. Other is “biased” which lands heads 75% of time. Which gives more info, fair or biased?  **Fair gives more info- higher info entropy b/c 50% chance instead of 75%/25%.**  What would this print?  D = { }  D[1] = ‘a’  D[2] = ‘b’  D[‘c’] = 3  Print ‘a’ in d.values()  **True**  After following is executed, what is type of variable s?  S = “This is some text”.split()  **List** | **List deletion:**  Ex: del list[x:y] 🡪 deletes items from x to y in list  **List methods:**  .append – adds items, end of list  .insert – inserts items to list  .reserve – reverses items in list  .sort – sorts items in list  .remove – removes items  **Split:**  Breaks string into list of words  *Ex: x=”Hey how are you” x.split() = [‘Hey’, ‘how’, ‘are’, ‘you’]*  Takes out what you’re splitting at  **Iteration:**  For loop – iterates over each element of statement  *Ex: for x in “Hello”:*  *Print x 🡪 prints each element on separate lines*  *Ex: fruit [“apples”, “bananas”]*  *For x in fruit:*  *Print x 🡪 prints each element of list on separate lines*  **Accumulator:**  *Ex: nums = [1, 2, 3]*  *Accum = 0*  *For x in accum:*  *Accum = accum + x*  *Print accum 🡪 returns 6, adds values in list starting at 0*  **Data files:**  Open & read – open(filename, ‘r’)  Open & write – open(filename, ‘w’)  Close file – filevariable.close()  Read line – filevar.readline()  Returns list of strings – filevar.readlines()  **Dictionaries:**  Keys – returns view of keys in dictionary  Values – return values in dictionary associated w/ keys  Items – return key-value pairs from dictionary  x.get – returns value associated with key  **Accumulate texts:**  Finding amount of occurrences in dictionary  *Ex: f = open(‘document.text’, ‘r’)*  *Txt = f.read()*  *T\_count = 0*  *For x in txt:*  *If x == ‘any letter’:*  *Anyletter\_count = anyletter\_count + 1*  *Print “any letter:” + str(anyletter\_count) + “occurences”*  **Function:**  Def name(paramters):  Return statements used in functions to return value  You’re connected to directory, ~/Documents/Courses/106/F14/Exams. Write unix command that displays contents of ~/Documents/Courses/106  **Cd ~/Documents/Courses/106 *or* ls ~/Documents/Courses/106/**  **Tuples:**  Represent records  Name = (“List”, “of”, “many”,“records”, “145”)  **While:**  Evaluates whether condition is true/false, if false, exits while statement, if true, executes each statement, then goes back to step 1  For following: L = [“First”, “Second”, “Third”]  What would print?  For x in L:  Y=L[0]  Print y  **First**  **First**  **First**  For x in L:  Y=L[0]  Print y  **First**  For x in L:  Y = x in L  Print y  **True**  **True**  **True**  What will this print?  X= 10.5  If x < 10:  Print “one”  Elif x < 20:  If x > 15:  Print “two”  Else:  Print “three”  Elif x > 0 # if x > 0  Print “four”  **Three**  #**Four**  What would this print?  L = []  L.append(‘a’)  L.append(‘b’)  L.append(‘c’)  L[1]=0  Print L  **[‘a’, 0, ‘c’]**  What would this print?  D={ }  D[1] = ‘a’  D[2] = ‘b’  D[‘c’]=3  D[‘c’] = d[‘c’]+1  Print d[‘c’]  **4**  Write statement to print 2nd element from list L to print e.  L=[‘h’, ‘e’, ‘l’, ‘l’, ‘o’]  **Print L[1]**  After following is executed, what is type of n[:3]  N=”abcde”  **String**  After following is executed, what is type of int(n[:3])  N=”abcde”  **Error**  What will following print?  Myvar=”hello”  Print myvar[-1]  **O**  What will following print out?  Myvar=”hello”  Print “myvar”[-1]  **R**  After following is executed, what is type of y?  Myvar=”the value of x is”  X=10  Y=myvar+x  **Error -----------------------------------**  Add until neg number  tot = 0  num = int(raw\_input("next num; -1 to stop> "))  while num >= 0:  tot = tot + num  num = int(raw\_input("next num; -1 to stop> "))  print tot | **Unix:**  Path to Desktop – Users/Name-of-Comp/Desktop  cd – change direction- cd /Users/Name/Desktop  Connect to home directory – cd ~  Connect to folder – cd Documents/Name-of-folder  ls – lists all files in a directory🡪 /Documents/Folder ls  cat – shows contents of a file/lets you concatenate them together  *Ex: cat sample.txt*  *To concatenate: cat sample1.txt sample2.txt*  less – lets you move between lines in a file  *Ex: less sample.txt*  q = quit  cd ../ - allows you to go back a layer  grep “program” a\_file – lists all files that contain “program”  | (pipe) – puts output from a file as the input on another  list1>list2 - puts list 1 in list2  python file.py – gets a python file  **Dictionaries ex:**  D = {“one”:1, “two”:2, “three”:3}  Print d.keys()  *Prints [‘one, ‘two’, ‘three’],* same thing happens with values  For x in d:  Print d[x], prints values,  1  2  3  print x - prints keys  **String methods:**  .upper – all caps  .lower – all lowercase  .count – returns # of occurrences  .find – returns left most index where substring is found, if not there, gives  -1  .index – like find but causes a runtime error if not found  Write 1 line of code that accomplishes what last three lines of code do  T = (20, 30, 40)  X = t[0]  Y = t[1]  Z = t[2]  **X, y, z = t**  What value prints?  Def g(x, y):  Z = y+x  Return y  Y=10  Z=g(5,y)  Print z  **10**  What will this print?  X = -1  Y = -2  Z = -3  Def h(x, y=2, z=3):  Print x, y, z  H(1)  **1, 2, 3**  Assume following has been executed  L = [{‘a’:1, ‘b’:2, ‘d’:11},  {‘a’:4, ‘b’:5, ‘e’:11},  {‘a’:7, ‘b’:8, ‘f’:11}]  Write code to print each value associated with key ‘b’ in each of dictionaries in L, should print  2  5  8  **for d in L:**  **print d[‘b’]**  days = {'sunday': 0, 'monday':1, 'tuesday':2, 'wednesday':3, 'thursday':4, 'friday':5, 'saturday':6}  x = raw\_input("enter day leaving")  day\_leave\_num = days[x]  trip\_length = int(raw\_input("enter how many nights are gone"))  total\_day = trip\_length + day\_leave\_num  day\_6 = total\_day%6  for vals in days.items():  if vals[1] == day\_6:  print vals[0]  Write code that generates a single dictionary with 1 key for each of keys in any of dictionaries in L, and value equal to count of how many dictionaries key appears in. Dictionary it generates should be:  {‘a’:3, ‘b’:3, ‘c’:3, ‘d’:1, ‘e’:1, ‘f’:1}  **dx = {}**  **for d in L:**  **for x in d.keys():**  **if k in dx:**  **dx[k] = dx[k] + 1**  **else:**  **dx[k] = 1**  or  **def new\_dict(lst):**  **dx={}**  **for d in lst:**  **for k in d.keys():**  **if k in dx:**  **dk[k] = dk[k]+1**  **else:**  **dx[k]=1**  **return dx**  **new\_var=new\_dict(L)**  Write code that repeatedly asks user to input numbers. Keep going until sum is 21 or more. Print out total.  **Sum = 0**  **While sum < 21:**  **X = int(raw\_input(“Please enter a number”))**  **Sum = sum + x**  **Print sum**  After following is executed, what is type of variable t?  S=”2014-10-02T20:12:28+0000”  T=split(“-“)  **List, error**  After following is executed, what is type of variable t?  S=”<published>2009-01-23T20:04:53z</published>”  T=s.find(“2009”)  **Integer**  What will print?  S=”2014-10-02T20:12:28+0000”  Print s[s.find(“14-“)]  **1**  What will print?  S=”2014-10-02T20:12:28+0000”  Print len(s.split(“:”)[2].split(“2”))  **2**  2 files, 1 has full poem, other has compressed version using lossless compression. Which has more info or do they have the same?  **Same- can reconstruct 1 from other bc lossless compression. 1 requires more storage than the other but same amount of info.**  Comp program convinces judges that it is human. Would Searle say program was “intelligent”?  **No, behaving intelligently does not equal understanding.**  Unix is run, 1 is 1000 lines, 2 is 2000 lines, nothing in common, how many lines in file3.txt?  Diff file1.txt file2.txt > file3.txt  **About 3000**  Define function scrabble\_score. Takes a word as input and takes a dictionary that letters=number scores. Should return an integer.  **Def scrabble\_score(word, vals\_dict):**  **Tot=0**  **For c in word:**  **Tot += vals\_dict[c]**  **Return tot---------------------------------**  current\_time = int(input("What is the current time (in hours)? "))  waiting\_time = int(input("How many hours do you have to wait? "))  hours = current\_time + waiting\_time  timeofday = hours % 12  print timeofday | Function that is\_prefix, takes 2 strings – true if 1st is prefix of second, false otherwise  Def is\_suffix(x, y):  A = len(x)  If x in y[a:]:  Return True  Else:  Return false  **Print is\_suffix(“He”,”Hello”)**  **Print is\_suffix(“Hi”, “Hello”)**  **Print is\_sffix(“lo”, “Hello”)**  x = open(“file.txt”, “r”)  z = x.read()  a = z.split(“”)  book\_dict={}  for b in a: #to do word  for d in b # to do letter  if b in book\_dict: #b to d, letter  book\_dict[b] +1 #b to d, letter  else:  book\_dict[b] =1 #b to do, letter  most = 0  most\_key = “”  for c in book\_dict:  if book\_dict[c] > most:  most = book\_dict[c]  most\_key = c  print most, most\_key --------------  For loop to print 2nd element of each tuple  **New\_tuple\_list = [(1,2), (4, “word”), (“hi”, “hey”), (“soda”, 5.2”)]**  **For h in new\_tuple\_list:**  **Print h[1:2]**  User input until user enters “quit”  **L = “”**  **While l != “quit”:**  **L = raw\_input(“Enter text. To stop, enter ‘quit.”)**  Define function takes list of integers as input, returns integer with max abs value.  Def abs(x):  If x < 0:  Return –x  Else:  Return x  Import test  Text.testEqual(maxabs([-5, 2])  **Def maxabs(L):**  **M = L[0]**  **For x in L:**  **If abs(x) > abs(m):**  **M=x**  **Return m**  How to make list of all guesses in dictionary  **Res = []**  **For k in d:**  **For let in d[k][“guesses”]:**  **Res.append(let)**  Write code to count # of strings that have w  Items = [“whirring”, “this”, “wry”]  **Acc\_num = 0**  **For x in items:**  **If ‘w’ in x:**  **Acc\_num = acc\_num+1**  **Print acc\_num** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Write code to print each element and type  Things = [“hello”, 2, 6.5]  **For w in several\_things**  **Print w**  **For w in several\_things:**  **Print type(w)**  What will print?  X=3  Y=4  X==y+1  Print x  **3**  Define function called deduplicate that takes a list as input and returns a list that has all duplicates removed, keeping only first instance of each item.  Ex: if [1, 2, 3, 2, 4, 2, 3, 4, 5], would return [1, 2, 3, 4, 5]  **Def deduplicate(lst):**  **Acc\_list = []**  **For x in lst:**  **If x not in acc\_lst:**  **Acc\_lst.append(x)**  **Return acc\_lst**  After following is execuded, what is type of x?  X = “The answer is “ + 42  **None of the above; error**  Will following cause run-time error? If not, what will print?  W = “Hi”  X = w == “Hello there”  Print(x)  **No error. Prints False**  Which are tests that authenticate someone as a human or \*very\* human-like computer, meaning similar in spirit to Turing Test?  *CAPTCHA puzzle that asks website visitor to transcribe blurred image of a word*  **Humans can, computers can’t.**  *Security question on website that asks your moms maiden name*  **No- authenticates that you’re a particular human, not that you’re human rather than computer.**  *Task listing all state capitals*  **No. Computers can do this.**  Consider following 2 statements. Which has higher entropy?  *I ate a big breakfast this…*  *I wonder if…*  **Second has higher because more possibilities for rest of sentence.**  Following code  St = “thank you, be right back”  Y = st.split()  What prints?  For w in y:  Print w[0]  **T**  **Y**  **B**  **R**  **B**  Write 3 function calls to function give\_greeting to get Hello, SI106!!!, Hello, world!!!, Hey, everybody!  Defgive\_greeting(greet\_word=”hello”, name=”SI106”, num\_exclam=3):  Final\_string=greet\_word+”,”+name+!”\*num\_exclam  Return final\_string  **Give\_greeting()**  **Give\_greeting(name=”world”)**  **Give\_greeting(“Hey”, “everybody”, 1)**  **Print *each one*­­**  What prints?  L=[]  L.append(‘a’)  L.append(‘b’)  L.append(‘c’)  L[‘a’]=0  Print L  **Error--------------------------------**  txt = raw\_input("enter name (palidrome checker as well)")  print txt[::-1] #EtAn  print txt[1:2] #A  print txt.upper() #NATE  print txt.lower() #nate  txt\_backward = txt[::-1]  if txt\_backward == txt:  print True #prints if works | Function that inputs 2 integers and returns multiplied value  **Def mult\_both(a=3, b=4):**  **Return a\*b**  **Print mult\_both()**  Function takes input and returns # of vowels  **Def get\_values(s):**  **Vowels=”aeiou”**  **Total = 0**  **For v in vowels:**  **Total+=s.count(v)**  **Return total**  **Print get\_vowels(“Hello all”)**  Common word that takes string/prints tuple of most common word and # of times word is used  Common\_word(“hello hello is what they said!”)  **Def common\_words(s):**  **D={}**  **Sp=s.split()**  **For w in sp:**  **If w in d:**  **D[w]=d[w]+1**  **Else:**  **D[w] = 1**  **Kys = d.keys()**  **Most\_common=kys[0]**  **For k in d:**  **If d[k] > d[most\_common]:**  **Most\_common = k**  **For ky in d:**  **If d[ky] == d[most\_common]:**  **Print ky, d[ky]**  Code that takes dictionary with key-val pairs and returns name with lowest value  Df = {“Nick”:56, “Paul”:73, “Jackie”:42}  **Def small\_val\_name(d):**  **Kys = d.keys()**  **M=kys[0]**  **For k in kys:**  **If d[k] < d[m]:**  **M=k**  **Return m**  Function that takes list of integers or string and returns sum  **Def sum\_a\_list\_or\_digitstring(lt):**  **Tot=0**  **For i in lt:**  **Tot=tot+int(i)**  **Return tot**  **Print sum\_a\_list(“1,4,7,5”)**  Following is executed  L=[“First”, “Second”, “Third”]  For x in L:  Y=x[0]  Print y  **T**  For L in x:  Print L[0]  **Error**  For x in L:  Print x[2] in L[0]  **True**  **False**  **True**  What prints?  X=7  If x<10:  Print “one”  If x<20:  If x>15:  Print “two”  Else:  Print “three”  Elif:  **One**  **Three**  What will print?  X=27  If x<10:  Print “one”  If x<20:  If x>15:  Print “two”  Else:  Print “three”  Elif x<30:  Print “four”  **Four**  What prints?  D={}  D[1]=’a’  D[2]=’b’  D[‘c’]=3  Print ‘a’ in d  **False**  What prints?  D={}  D[1]=’a’  D[2]=’b’  D[‘c’]=3  Print d[1]+d[2]  **ab**  Write 1 line of code that does what last 3 lines do  X=”some string”  Y=20  Z=x  X=y  Y=z  **(z, x, y)=(x, y, z)**  What prints?  X=-1  Y=-2  Z=-3  Def h(x, y=2, z=3):  Print x, y, z  H(1,4)  **1 4 3**  Following has been executed  L=[{‘a’:1, ‘b’: 2, ‘d’:11}, {‘a’:4, ‘b’:5, ‘d’:12}, {‘a’:7, ‘b’:8, ‘f’:11}]  Write code to print each value associated with key d  **For diction in L:**  **If ‘d’ in diction:**  **Print diction[‘d’]**  Write code that creates 1 dictionary with 1 key for any key that appears in L with value of sum of keys  **Sums = {}**  **For diction in L:**  **For k in diction:**  **If k in sums:**  **Sums[k] += diction[k]**  **Else:**  **Sums[k]=diction[k]**  Write code that asks user to input numbers and keeps going until > 21  **Sum=0**  **Nums=[]**  **While sum < 21:**  **Nxt=int(raw\_input(“Insert a number”)**  **Sum += nxt**  Nums.append(nxt)  **Print nums-------------------------**  prefixes = "JKLMNOPQ"  suffix = "ack"  special = ("O", "Q")  for x in prefixes:  if x in special:  print x + "u" + suffix  else:  print x + suffix |