i116: Basic of Programming

3. Built-in data structures

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Strings
- Tuples
- Lists
- Dictionaries (or Maps)
- A billing program

Strings

- What is enclosed with single quotes '....'.
- Let *s* be 'JAIST'.
  - *s*[0] returns 'J'.
  - *s*[1] returns 'A'.
  - *s*[5] raises an exception called IndexError.
- + is used to concatenates strings.
  - Let *s1*, *s2*, *s3*, *s4*, *s5*, *s6*, and *s7* be 'Japan ', 'Advanced ', 'Institute ', 'of ', 'Science ', 'and ', and 'Technology'.
  - *s1*+*s2*+*s3*+*s4*+*s5*+*s6*+*s7* returns the following:

  ‘Japan Advanced Institute of Science and Technology'

Strings

- Writing a long string, we may want to use multiple lines.
- When this is the case, we need to use a backslash \ just before each carriage return (CR), e.g.,

```
'Japan '\
'Adcanced '\
'Institute '\
'of '\
'Science '\
'and '\
'Technology'
```

Strings

- A tab can be used in a string and should be written as \t in it.

  'Japan\tAdvaneced\tInstitute\tof\tScience\tand\tTechnology'

- When you would like to use a single quote, you should write a backslash \ just before it.

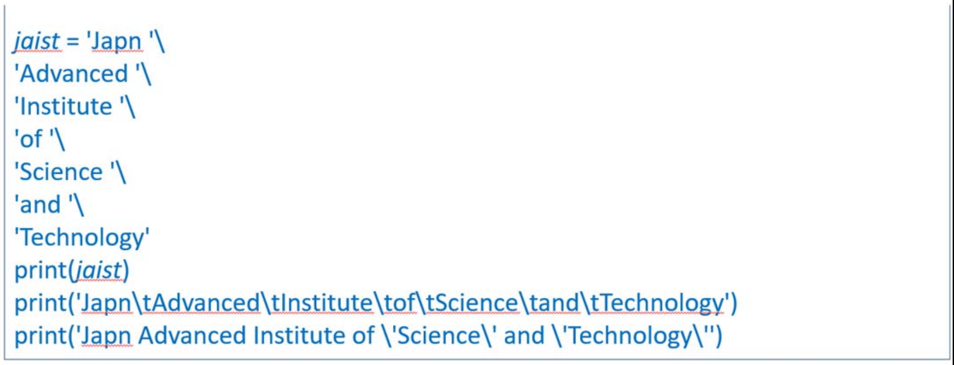
  'Japan Advanced Institute of \'Science\' and \'Technology\''

A backslash, a single quote, and a single quote.

Strings

```
s = 'JAIST'
print(s[0])
print(s[1])
try:
   print(s[5])
except IndexError as em:
   print(em)
s1 = 'Japn '
s2 = 'Advanced '
s3 = 'Institute '
s4 = 'of '
s5 = 'Science '
s6 = 'and '
s7 = 'Technology'
print(s1+s2+s3+s4+s5+s6+s7)
```

Strings



```
jaist = 'Japn '\
'Advanced '\
'Institute '\
'of '\
'Science '\
'and '\
'Technology'
print(jaist)
print('Japn\tAdvanced\tInstitute\tof\tScience\tand\tTechnology')
print('Japn Advanced Institute of \'Science\' and \'Technology\'')
```

Tuples

$(e_0, e_1, \dots, e_i, \dots, e_N)$ or $(e_0, e_1, \dots, e_i, \dots, e_N,)$

- A collection of data; the order is relevant.
- The final comma must be written when $N = 0$; e.g., ('zero',) is a tuple, but ('zero') is not and a string, the same as 'zero'.
- The final comma must not be written when you would like to write the empty tuple (); (,) cannot be accepted.
- Can consists of different types; e.g., (0, 'zero', 0.0) consists of the three elements whose types are int, string and float.
- When $N = 1$, it is a pair (e_0, e_1); when $N = 2$, it is a triple (e_0, e_1, e_2).

Tuples

Let *tpl* be $(e_0, e_1, \dots, e_i, \dots, e_N)$.

- *tpl*[*i*] returns e_i.

For example, *tpl*[0] and *tpl*[1] return e_0 and e_1.

- *tpl*[-*j*] returns the *j*th element from the bottom.

For example, *tpl*[-1] and *tpl*[-2] return e_N and e_{N-1}.

- If *k* is out of the range, such as $N + 1$, *tpl*[*k*] raises an exception called IndexError.

Tuples

Let *tpl* be $(e_0, e_1, \dots, e_i, \dots, e_N)$.

- *tpl*[*i*] cannot be updated with an assignment.

tpl[*i*] = e'_i causes an exception called TypeError.

- If you really want to change e_i to e'_i, one possible way to do is as follows:

  tpl = (*tpl*[0], *tpl*[1], …, e'_i, …, *tpl*[*N*])

making a new tuple all of whose elements are the same as those in (the old version of) *tpl* except for the ith one.

Tuples

```
aTuple = (0, 'zero', 0.0)
print(aTuple)
print((), ' is the empty tuple.')
print(('zero',), ' is the tuple that only consists of \'zero\'.')
print('(\'zero\') is not a tuple but a string, the same as \'zero\'.')
print('(\'zero\') == \'zero\' returns ', ('zero') == 'zero', '.')
print('(\'zero\',) == \'zero\' returns ', ('zero',) == 'zero', '.')
print(aTuple[0])
print(aTuple[1])
print(aTuple[-1])
print(aTuple[-2])
```

Tuples

```
try:
   aTuple[2] = 1.41421356
except TypeError as em:
   print('If we try to do aTuple[2] = 1.41421356, the following message is written:')
   print(em)
```

```
try:
   aTuple[3]
except IndexError as em:
   print('If we try to do aTuple[3], the following message is written:')
   print(em)
```

```
aTuple = (aTuple[0], aTuple[1], 1.41421356)
print(aTuple)
```

Lists

$$[e_0, e_1, \dots, e_i, \dots, e_N]$$

- A collection of data; the order is relevant.
- Can consists of different types; e.g., [0, 'zero', 0.0] consists of the three elements whose types are int, string and float.
- It would be, however, better to have values of one type in a list, such as [0, 1, 2, 3].
- When you want to use a collection of data whose types are different, you should use a tuple.

Lists

Let *lst* be $[e_0, e_1, \dots, e_i, \dots, e_N]$.

- *lst*[*i*] returns e_i.

For example, *lst*[0] and *lst*[1] return e_0 and e_1.

- *lst*[-*j*] returns the *j*th element from the bottom.

For example, *lst*[-1] and *lst*[-2] return e_N and e_{N-1}.

- If *k* is out of the range, such as $N + 1$, *lst*[*k*] raises an exception called IndexError.
- *lst*[*i*] = e'_i updates the ith element to e'_i from e_i.

Lists

```
aList = [0,1,2,3,4]
print(aList)
print(aList[0])
print(aList[1])
print(aList[-1])
print(aList[-2])
aList[2] = 10
print(aList) # aList[2] = 10 changes aList.

try:
   aList[5]
except IndexError as em:
   print('If we try to do aList[5], the following message is written:')
   print(em)
```

Lists

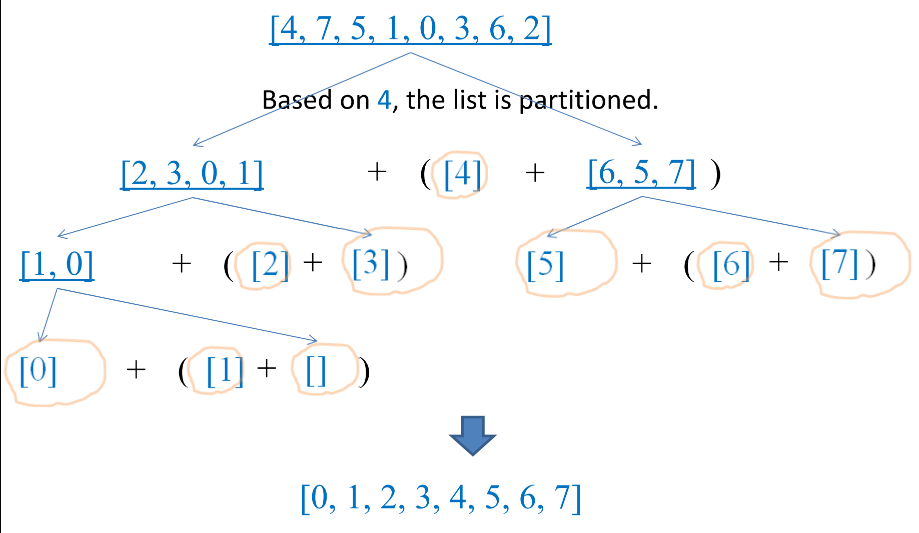
- *lst*[*x*:*y*] extracts a sub-list from *lst* (or slices *lst*).
  - If *lst* is $[e_0, \dots, e_x, \dots, e_{y-1}, e_y, \dots, e_N]$, *lst*[*x*:*y*] is $[e_x, \dots, e_{y-1}]$, and *lst*[*y*:*x*] is [ ].
- *lst*[*x*:] also extracts a sub-list from *lst*, and so does *lst*[:*y*].
  - If *lst* is $[e_0, \dots, e_x, \dots, e_{y-1}, e_y, \dots, e_N]$, *lst*[*x*:] is $[e_x, \dots, e_y, \dots, e_N]$ and *lst*[:y] is $[e_0, \dots, e_x, \dots, e_{y-1}]$.
- *lst*[*x*:*y*], *lst*[*x*:] and *lst*[:y] do NOT alter *lst*.
- Let *lst2* be a list as well; *lst* + *lst2* is the list obtained by concatenating the two lists in this order; *lst* + *lst2* does NOT alter *lst* nor *lst2*.

Lists

```
print(aList[1:4])
print(aList[2:1])
print(aList[1:]) # deleting the top element
print(aList[:-1]) # deleting the bottom element
print(aList[100:])
print(aList[:-100])
print(aList) # aList[1:4] ... do not change aList.
print(aList[-100:100]) # seems strange but returns the list stored in aList
print(aList + aList)
print(aList) # + does not change aList.
print([-1] + aList)
print(aList + [5])
```

Lists

Sorting with Quicksort



Lists

```
def qsort(lst):
  if len(lst) <= 1:
    return lst
  else:
    pair = partition(lst[0],lst[1:])
    return qsort(pair[0]) + [lst[0]] + qsort(pair[1])
```

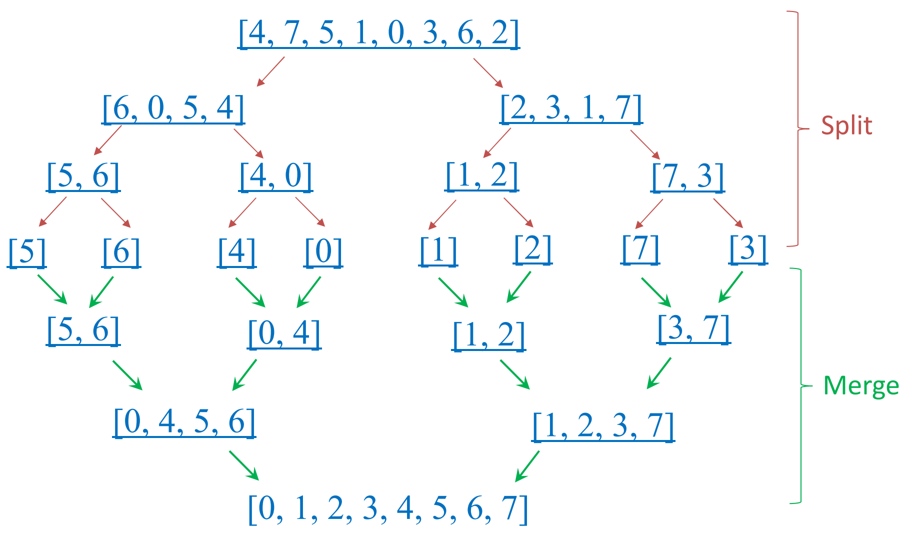
Lists

```
def partition(pvt,lst):
  pair = ([], [])
  for e in lst:
    if e < pvt:
      pair = ([e] + pair[0], pair[1])
    else:
      pair = (pair[0], [e] + pair[1])
  return pair
```

```
lst = [4,7,5,1,0,3,6,2]
print('Input: ', lst)
print('Output: ', qsort(lst))
```

Lists

Soring with Mergesort



Lists

```
def msort(lst):
    if len(lst) <= 1:
        return lst
    else:
        pair = split(lst,[],[])
        return merge(msort(pair[0]),msort(pair[1]))
```

```
def split(lst,l1,l2):
    if len(lst) == 0:
        return (l1,l2)
    else:
        return split(lst[1:],l2,[lst[0]]+l1)
```

Lists

```
def merge(l1,l2):
  if len(l1) == 0:
    return l2
  elif len(l2) == 0:
    return l1
  else:
    if l1[0] < l2[0]:
      return [l1[0]] + merge(l1[1:],l2)
    else:
      return [l2[0]] + merge(l1,l2[1:])
```

```
lst = [4,7,5,1,0,3,6,2]
print('Input: ', lst)
print('Output: ', msort(lst))
```

Dictionaries

- Terminology “*association list*” or “*a-list*” used in AI.
- Terminology “*map*” used in Java, etc.
- Terminology “*dictionary*” used in Smalltalk, Python, etc.
- Basically, a collection of (key, value)-pairs such that the value associated with a key can be retrieved.

Dictionaries

$$\{k_0: v_0, k_1: v_1, \ldots, k_i: v_i, \ldots, k_N: v_N\}$$

- k_i is a key and v_i is the value associated with k_i.
- The order is not relevant; e.g., {'x':1.41,'z':1.73} and {'z':1.73, 'x':1.41} are the same.

Let *aDict* be the dictionary.

- *aDict*[k_i] returns v_i if k_i is registered; otherwise, it raises an exception called KeyError.
- *aDict*[k_i] = v'_i updates the value associated with k_i if k_i is registered; otherwise, it adds k_i and v'_i to *aDict*.

Dictionaries

```
aDict = {'x':1.41, 'y':3.14, 'z':1.73}
aDict2 = {'y':3.14, 'z':1.73, 'x':1.41}
print(aDict)
print(aDict2)
print(aDict, ' == ', aDict2, ' returns ', aDict == aDict2, '.')
print(aDict['x'])
print(aDict['z'])
```

```
try:
   print(aDict['a'])
except KeyError as em:
   print('If we do aDict[\'a\'], we have the follwing message:')
   print(em)
```

Dictionaries

```
aDict['a'] = 2.71
print(aDict)
print(aDict['a'])
aDict['x'] = 2.23
print(aDict)
```

Dictionaries

```
aDict = {'x':1.41, 'y':3.14, 'z':1.73}
aDict2 = {'y':3.14, 'z':1.73, 'x':1.41}
print(aDict)
print(aDict2)
print(aDict, ' == ', aDict2, ' returns ', aDict == aDict2, '.')
```

aDict equals *aDict2* as dictionaries, but may make the program behavior different.

```
x = 0
for k in aDict:
    x = x + 1
    if k == 'z':
        break
print('x = ', x)
```

```
x = 0
for k in aDict2:
    x = x + 1
    if k == 'z':
        break
print('x = ', x)
```

A billing program

A catalog {'mp':('MacPro', 5000000), 'im':('iMac', 400000), 'mbp':('MacBook Pro', 500000), 'am':('AirMac', 200000)}

A cart [('am', 4), ('mbp', 2), ('mp',1), ('am', 3), ('mp', 1)]

A bill ([('AirMac', 7, 1400000), ('MacBook Pro', 2, 1000000), ('MacPro', 2, 10000000)], 12400000)

We will be creating a program that makes a bill from a catalog and a cart.

A billing program

```
catalog ={'mp':('MacPro', 5000000), 'im':('iMac', 400000), 'mbp':('MacBook Pro', 500000), 'am':('AirMac', 200000)}
cart = [('am', 4), ('mbp', 2), ('mp',1), ('am', 3), ('mp', 1)]
```

```
def normCart(c):
  tc = []
  flg = True
  for i in range(len(c)):
    for j in range(len(tc)):
      if (c[i])[0] == (tc[j])[0]:
        tc[j] = ((tc[j])[0], (c[i])[1] + (tc[j])[1])
        flg = False
        break
    if flg:
      tc = tc + [c[i]]
    flg = True
  return tc
```

A billing program

```
print(normCart(cart))
```

```
def mkBillItemLst(cat,nc):
    bil = []
    for i in range(len(nc)):
        try:
            ip = cat[(nc[i])[0]]
            bil = bil + [(ip[0], (nc[i])[1], ip[1] * (nc[i])[1])]
        except KeyError:
            return []
    return bil
```

```
print(mkBillItemLst(catalog,normCart(cart)))
```

A billing program

```
def mkBill(cat,c):
    bil = mkBillItemLst(cat,normCart(c))
    ttl = 0
    for bi in bil:
        ttl = ttl + bi[2]
    return (bil, ttl)
```

```
print(mkBill(catalog,cart))
```

A billing program

```
def printBill(bll):
    bil = bll[0]
    ttl = bll[1]
    print('************** Billing **************')
    print('item ordered\t#items\tsub-total')
    for bi in bil:
        print(bi[0], '\t', bi[1], '\t', bi[2])
    print('*********** total amount ***********')
    print(ttl, ' Japanese Yen')
```

```
printBill(mkBill(catalog,cart))
```