National University of Singapore
School of Computing
CS1010X: Programming Methodology
Semester II, 2020/2021

Solutions for Recitation 8
Dictionaries & Message Passing

Python

1. *{}* - dictionary constructor

   By itself, creates an empty dictionary. Initialize with elements in this manner: `{key1:element1, key2:element2, ···, keyN:elementN}`

2. *dict* - `dict(`<*sequence of pairs*>`)`

   Takes in a *sequence* type consisting of sequence of pairs (tuples) and converts it into a dictionary. If no sequence is provided, i.e. `dict()`, an empty dictionary is returned. If the provided sequence is not a sequence of pairs, this will cause an error.

3. Assignment - `<dict>[`*key*`] = `*value*. Assigns a new *value* to the specified *key* in the dictionary `<dict>`. This updates an existing record if one exists, and creates a new record if none exists.

4. Deletion:

   (i) `del <dict>[`*key*`]`. Deletes the record corresponding to the specified *key* in the dictionary `<dict>`, if one exists.

   (ii) `<dict>.clear()`. Remove all entries in `<dict>`.

   (iii) `del <dict>`. Deletes the dictionary `<dict>`.

5. Access:

   (i) `<dict>.get(key, default=None)`. For key *key*, returns value, or default if *key* is not in dictionary `<dict>`.

   (ii) `key in <dict>`. Returns `True` if *key* in dictionary `<dict>`, `False` otherwise.

   (iii) `<dict>.keys()`. Returns list of dictionary `<dict>`'s keys.

   (iv) `<dict>.values()`. Returns list of dictionary `<dict>`'s values.

   (v) `<dict>.items()`. Returns a list of `<dict>`'s (key, value) tuple pairs

   (vi) `len(<dict>)`. Returns the number of elements in `<dict>`.

Problems

1. Evaluate the following expressions:

```
a = (("apple", 2), ("orange", 4), (5, 7))
b = dict(a)

c = [[1, 2], [3, 4], [5, 7]]
d = dict(c)

print(b["orange"])   #=> 4

print(b[5])   #=> 7

print(b[1])   #=> KeyError

b["bad"] = "better"
b[1] = "good"

for key in b.keys():
    print(key)

#=> apple
#=> orange
#=> 5
#=> bad
#=> 1

for val in b.values():
    print(val)

#=> better
#=> good
#=> 4
#=> 7
#=> 2

del b["bad"]
del b["apple"]

print(tuple(b.keys()))    #=> (1, 'orange', 5)

print(list(b.values()))   #=> ['good', 4, 7]
```

2. **Stack Implementation (in Message-Passing Style).** Implement a stack object with the following functions:

   (i) `make_stack`: returns a new empty stack object.

   (ii) `s("is_empty")`: returns `True` if the stack `s` is empty.

   (iii) `s("clear")` : empties the stack `s` of any elements it may contain.

   (iv) `s("peek")`: returns the top element of the stack `s`, leaving the stack unchanged. If the stack is empty, returns `None`.

   (v) `s("push")(item)`: pushs an element `item` onto the top of the stack `s`.

   (vi) `s("pop")`: removes and returns the top element of the stack `s`. If the stack is empty, returns `None`.

   Sample execution:

```
s = make_stack()
print(s("is_empty")) # True
s("push")(1)
s("push")(2)
print(s("peek"))     # 2
print(str(s("pop"))) # 2
print(str(s("pop"))) # 1
print(str(s("pop"))) # None

def make_stack():
    items = []
    def oplookup(msg):
        if msg == "is_empty":
            return len(items)==0
        elif msg == "clear":
            items.clear()
        elif msg == "stuff":
            return items
        elif msg == "peek":
            if len(items)==0:
                return None
            else:
                return items[len(items)-1]
        elif msg == "push":
            def push_op(item):
                items.append([item])
            return push_op
        elif msg == "pop":
            if len(items)==0:
                return None
            else:
                return items.pop()
        else:
            raise Exception("stack doesn't" + msg)
    return oplookup
```

3. Write a function called `push_all` which takes a stack and a sequence and pushes all the elements of the sequence onto the stack. It should return the stack.

```
def push_all(s, tup):
    for item in tup:
        s("push")(item)
    return s
```

4. Write a function called `pop_all` which takes a stack and pops elements off it until it becomes empty, adding each element to an output list.

```
def pop_all(s):
    result = []
    while (not s("is_empty")):
        result.append(s("pop"))
    return result
```

5. **(Homework) Calculator Object Implementation**

```
c = make_calculator()
c('ANSWER')                    # empty_stack
c('NUMBER_INPUT')(4)           # pushed
c('ANSWER')                    # 4
c('NUMBER_INPUT')(5)           # pushed
c('ANSWER')                    # 5
c('OPERATION_INPUT')('+')      # pushed
c('ANSWER')                    # 9
c('NUMBER_INPUT')(7)           # pushed
c('OPERATION_INPUT')('-')      # pushed
c('ANSWER')                    # 2
c('CLEAR')                     # cleared
c('ANSWER')                    # empty_stack
```

   (i) Complete the definition of `oplookup` so it is a function that when given an operation name and the `ops` list, will return the operation with the given name.

   (ii) Write a method called `ANSWER`, which returns the current value on the top of the stack.

   (iii) Write a method called `CLEAR`, which removes all the numbers from the stack.

   (iv) Write a method called `NUMBER_INPUT`, which puts the number onto the stack.

   (v) Write a method called `OPERATION_INPUT`, which takes an operation name as input, looks up the operation, removes two numbers from the stack, and puts the result of the operation back onto the stack.

```
def make_calculator():
    stack = make_stack()
    ops = {'+':lambda x, y: x + y,
           '-':lambda x, y: x - y,
           '*':lambda x, y: x * y,
           '/':lambda x, y: x / y}
    def oplookup(msg):
        if msg == 'ANSWER':
            if stack("is_empty"):
                return "empty_stack"
            else:
                return stack("peek")
        elif msg == 'NUMBER_INPUT':
            return stack("push")
        elif msg == 'OPERATION_INPUT':
            def push_op(x):
                val1 = stack("pop")
                val2 = stack("pop")
                stack("push")(ops[x](val2,val1))
            return push_op
        elif msg == 'ANSWER':
            return stack("peek")
        elif msg == 'CLEAR':
            stack("clear")
        else:
            raise Exception("calculator doesn't" + msg)
    return oplookup
```