National University of Singapore
School of Computing
CS1010X: Programming Methodology
Semester II, 2024/2025

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"])

print(b[5])

print(b[1])

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

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

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

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

print(tuple(b.keys()))

print(list(b.values()))
```

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
```

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.

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.

5. **Calculator Object Implementation**

```
def make_calculator():  #an RPN 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, *args):
        # YOUR CODE BEGINS HERE


        # YOUR CODE ENDS HERE
        else:
            raise Exception("calculator doesn't" + msg)
    return oplookup


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