Tentamen för TDP002 Imperativ programmering - Python
2010-10-22 08-12

Ola Leifler

Instruktioner:

- Skriv svar på varje fråga i en separat fil i hemkatalogen (~) namngiven enligt mönstret `login-uppgift.(txt)|(py)`, exempelvis `olale507-1.txt` eller `fregy123-2.py`.
- Uppge namn och personnummer i varje inlämnad fil.
- Uppgifterna löses enskilt.
- Kursboken *Learning Python* är tillåten.
- Kommunikation med andra under tentamenstillfället är förbjudet.

Fråga:	1	2	3	4	Totalt
Poäng:	2	4	3	5	14

För betyg 3 gäller 50% rätt, för betyg 4 65% och för betyg 5 80%, avrundat till närmaste hela poäng.
Både rätt innehåll och välformulerad text krävs för full poäng på uppgifterna.
Det totala betyget på tentamen är detsamma som det lägsta betyget av teoridel och Pythondel. Det är lika många poäng på bägge delarna.

1. (2 poäng) Nedanstående kod återfinns i `given_files/q1.py`. Förklara vad som gör att `spam_synonyms` får två olika värden efter att uttrycken på raderna 4–6 evaluerats respektive uttrycken på raderna 9–11.

```
1  egg_synonyms = ['ovum', 'germ cell']
2  spam_synonyms = ['pickled ham']
3
4  python_terms1 = {'egg' : egg_synonyms,
5                   'spam': spam_synonyms+['canned beaf']}
6  python_terms1['spam'] += ['unsolicited commercial e-mail']
7  print spam_synonyms
8
9  python_terms2 = {'egg' : egg_synonyms,
10                  'spam': spam_synonyms}
11 python_terms2['spam'] += ['unsolicited commercial e-mail']
12 print spam_synonyms
```

Lösningsförslag: Efter raderna 4–6 ändras inte listan `spam_synonyms` då värdet på platsen med index `'spam'` i tabellen `python_terms1` är resultatet av `spam_synonyms+['canned beaf']`, vilket är en *ny* lista. Efter raderna 9–11 ändras däremot listan som `spam_synonyms` refererar till då `python_terms2['spam']` refererar till *samma lista*.

2. (4 poäng) Komplettera den abstrakta datatypen `time_period` nedan med det som behövs för att definiera funktionen `overlaps` som ska returnera `True` om `time_period1` överlappar `time_period2` och `False` annars. Bifoga testfall som visar att din funktion fungerar. För full poäng krävs att lösningen är så kort som möjligt med avseende på de jämförelser som görs mellan `time_period1` och `time_period2` samt att du implementerat och i `overlaps` använt funktioner för att hantera datatypen `time_period`.

```
# Tidsperiod
def create_time_period(start,end):
    return ('time_period', start, end)

# Överlappar tidsperioder?
def overlaps(time_period1, time_period2):
    pass
```

Lösningsförslag:

```
def create_time_period(start, end):
    return ("time_period", start,end)

def get_start(time_period):
    return time_period[1]

def get_end(time_period):
    return time_period[2]

# We can assume that the start and end times are comparable using <,
# >, <= ...

def overlaps(time_period1, time_period2):
    if (get_start(time_period1) > get_start(time_period2)):
        return overlaps(time_period2,time_period1)
    # time_period1 is ensured to begin no later than time_period2 does
    else:
        return get_end(time_period1) >= get_start(time_period2)

tp1=create_time_period(1,4)
```

```
tp2=create_time_period(2,6)
tp3=create_time_period(2,3)
tp4=create_time_period(5,8)

print "tp1 and tp2 should overlap: ",(overlaps(tp1,tp2) and overlaps(tp2,tp1))
print "tp2 and tp3 should overlap: ",(overlaps(tp2,tp3) and overlaps(tp3,tp2))
print "tp1 and tp4 should NOT overlap: ",(overlaps(tp1,tp4) or overlaps(tp4,tp1))
```

3. (3 poäng) Nedanstående funktion (som också återfinns i `given_files/q4.py`) ska konvertera decimala heltal till tal med basen 2, 3, 4, 5, 6, 7, 8 eller 9. Den fungerar dock inte riktigt som avsett. Se till att den fungerar korrekt och visa att den gör det.

Lösningsförslag:

```
import random

def convert_number(decimal_number,new_base):
    """ new_base should be in [2..9]
    """
    if not (new_base in range(2,10)):
        return "Not a valid base: %d"%new_base
    else:
        factors = []
        factor = decimal_number
        while factor >= new_base:
            # The *factor* should be divided by new_base, not decimal_number
            factor, rest = divmod(factor,new_base)
            factors = [rest]+factors
        # Last, we need to add the last factor to the list of factors
        factors = [factor]+factors
        print "%d in base %d: "%(decimal_number,new_base),
        resultstring = ""
        for factor in factors:
            resultstring+=str(factor)
        print resultstring
        return resultstring

def test_convert_numbers():
    # Test a few random numbers and see if the results are similar to
    # using 'bin' and 'oct'
    for i in range(10):
        num=int(random.random()*100)
        if (convert_number(num,2)!=bin(num)[2:]) or (convert_number(num,8)!=oct(num)
            [1:]):
            return "Not working for %d"%num
    return "10 samples working for bases 2 and 8"
```

4. I filen `given_files/q5.py` finns det en funktion `cmd_loop` som läser och utför kommandon som användaren anger. Ni ska utöka denna på två sätt:
   (a) (2 poäng) Se till att man kan använda variabler i uttryck, så att nedanstående exempel fungerar:

```
>>> let a 3
3
>>> mul a 4
12
```

(b) (3 poäng) Modifiera och utöka koden så att man med hjälp av en annan funktion `add_operation` kan lägga till nya operationer till `cmd_loop`. Funktionen `add_operation` ska ta två parametrar: namnet på en operation och en definition av operationen. För full poäng krävs att samtliga operationer som känns igen i `cmd_loop` ska vara sådana som ni lagt till med hjälp av `add_operation`.

Lösningsförslag: Det finns ett antal sätt som körexemplet misstolkats:

1. Variabler behöver endast förekomma som argument till `mul`.
2. Variabler behöver endast ha namnet `a`.
3. Variabler behöver endast kunna förekomma som det första argumentet till en funktion.
4. `mul` och `sum` kan begränsas till att endast acceptera 2 argument, jämfört med godtyckligt antal tidigare.

Jag har inte givit avdrag för fel 3 och 4, däremot ger misstolkningarna 1 och 2 poängavdrag.

```
def cmd_loop():
    # del a: slå upp variabler
    def resolve(x):
        if x in variables:
            return variables[x]
        elif x.isdigit():
            return int(x)
        else:
            return x
    # del b: add_operation
    def add_operation(op_name,op):
        commands[op_name]=op

    def assign_value(values):
        variables[values[0]]=values[1]

    commands = {}
    # del b: alla operationer ska läggas till med add_operation
    add_operation('sum',lambda values: reduce(lambda x,y:x+y,values))
    add_operation('mul',lambda values: reduce(lambda x,y:x*y,values))
    add_operation('mod',lambda values: values[0] % values[1])
    # Cannot do assignment in lambda, so use assign_value
    add_operation('let',assign_value)
    should_quit = False
    variables = {}
    quit_commands = ['quit', 'exit', 'bye']
    while not should_quit:
        cmd = raw_input(">>> ")
        should_quit=cmd in quit_commands
        cmd = cmd.split(" ")
        op_name = cmd[0]
        args = cmd[1:]
        values=map(lambda arg: resolve(arg),args)
        if op_name in commands:
            print commands[op_name](values)
        elif (op_name in variables):
            print variables[op_name]
        else:
            if (not should_quit):
                print "Unknown command: ", cmd
    print "Bye bye"

cmd_loop()
```