Tentamen i Programmering DI2006, Del 1

Datum: 2024-01-08
Tid: 15.00–18.00
Ansvarig lärare: Eric Järpe (tel: 0729-77 36 26, email: eric.jarpe@hh.se)

Anvisningar

- Tillåtna hjälpmedel är
  - formelsamling (som är häftad till tentamenstexten)
  - miniräknare TI-30Xa (Texas Instruments)
  - skrivpapper
  - penna
  - suddigummi
  - linjal
  - frukt, fika
- Till varje uppgift finns angivet hur många poäng som maximalt utdelas för uppgiften.
- Tentamen består av två delar: **Del 1** och **Del 2**.
- Samtliga frågor i Del 1 ska besvaras i den svarstalong som är bifogad med tentamenstexten.
- Del 2 ges på fredag.
- Då programkod anges som svar ska den vara i så körbart skick som möjligt.
- Del 1 består av 22 frågor och här kan man maximalt få 30 poäng.
- För betyg 3 från Del 1 måste man ha minst 15 poäng.
- Resultatet från Del 1 kan bara ge godkänt med betyg 3 oavsett hur många poäng man har.
- För högre betyg måste man först ha godkänt på Del 1 och sedan skriva tillräckligt många poäng på Del 2.

LYCKA TILL!

Del 1

Flervalsfrågor

1. Vad kallas det talsystem där man använder siffror som betecknas $0, 1, 2, \ldots, 9, A, B, \ldots, F$? (1p)
   (a) Det optimala talsystemet
   (b) `numalpha`
   (c) `char`
   (d) `int`
   (e) Det decimala talsystemet
   (f) Det hexadecimala talsystemet
   (g) Inget av de ovanstående alternativen

---

2. Om en variabel tilldelas värdet `False` i Python så blir variabelns typ (1p)
   (a) `int`
   (b) `bool`
   (c) `char`
   (d) `float`
   (e) `string`
   (f) `binary`
   (g) Inget av de ovanstående alternativen

---

3. Om variabeln `conc3` är definierad

```
conc3 = lambda s : 3*s
```

   vad returneras då man skriver `conc3(7)` respektive `conc3("s")`? (1p)
   (a) `7` och `"s"`
   (b) `37` och `"3s"`
   (c) `21` och `"sss"`
   (d) `777` och `["s","s","s"]`
   (e) `10` och `"s*s*s"`
   (f) `7:3*7` och `"s":3*"s"`
   (g) Inget av de ovanstående alternativen

---

4. Om det står `def` i början av en rad beyder det att det som står indraget under denna rad definierar (1p)
   (a) en funktion
   (b) en klass
   (c) ett flödesschema
   (d) en algoritm
   (e) ett program
   (f) en modul
   (g) Inget av de ovanstående alternativen

---

5. Programmeringsspråket Python är (1p)
   (a) ett lågnivåspråk
   (b) ett relationsbaserat språk
   (c) maskinskod
   (d) ett objektorienterat, imperativt, funktionellt språk
   (e) ett rent databasspråk
   (f) en dialekt av Java
   (g) Inget av de ovanstående alternativen

---

6. Vilket värde får variabeln `a` om man i Python exekverar koden

```
a = 0
while a<3:
    a += round(2*((a+1)**(-1)),1)
```

(1p)
   (a) `1.3`
   (b) `3`
   (c) `3.1415927`
   (d) `3.2`
   (e) `4.0`
   (f) `1.4285e+4`
   (g) Inget av de ovanstående alternativen

---

7. Vad kallas den metod som i definitionen av en klass heter `__init__`? (1p)
   (a) mutator
   (b) initiative
   (c) inkapsling
   (d) magisk metod
   (e) polymorfism
   (f) konstruktor
   (g) Inget av de ovanstående alternativen

---

8. Om Sauron skriver koden

```
age = []
weight = age
for i in range(2):
    aw = input("Ange din ålder och vikt separerade med ett mellanslag: ")
    age.append(int(aw.split()[0]))
    weight.append(int(aw.split()[1]))
print((age[2]\%13)*(weight[2]//20)**2)
```

och exekverar den och på frågan om ålder och vikt svarar `12 21` respektive `19 91` så får han utskriften (1p)

(a) `0`

(b) `12`

(c) `96`

(d) `10000`

(e) `IndexError`

(f) `SyntaxError`

(g) Inget av de ovanstående alternativen

---

9. Vad svarar Python om man skriver `'hokus pokus'[::-2]`? (1p)

(a) `'sukop sukoh'`

(b) `'soso'`

(c) `'ksu hp'`

(d) `'skpskh'`

(e) `AttributeError`

(f) `TypeError`

(g) Inget av de ovanstående alternativen

---

10. Datatyperna `list`, `string` och `dictionary` är alla exempel på (1p)

(a) sequentials

(b) mutables

(c) immutables

(d) integers

(e) generators

(f) functions

(g) Inget av de ovanstående alternativen

---

11. En dictionary består av par av objekt. Vad kallas varje sådant par? (1p)
    (a) `key`
    (b) `value`
    (c) `item`
    (d) `tuple`
    (e) `list`
    (f) `element`
    (g) Inget av de ovanstående alternativen

---

12. Vad ska man skriva på den andra raden i koden

```
fib = [1]*2

    fib.append(fib[-1]+fib[-2])
print(fib[-1])
```

    för att utskriften ska bli `13` då man exekverar den? (1p)
    (a) `with fib[-1] as fib:`
    (b) `try:`
    (c) `if fib[-1]<12:`
    (d) `for i in fib:`
    (e) `while len(fib)<7:`
    (f) `return fib`
    (g) Inget av de ovanstående alternativen

---

13. Vad kallas programmering där man kan skapa variabler som tar argument som indata returnerar värden som utdata? (1p)
    (a) `imperativ programmering`
    (b) `funktionell programmering`
    (c) `lågnivåprogrammering`
    (d) `pseudokod`
    (e) `smart teknologi`
    (f) `lambdakalkyl`
    (g) Inget av de ovanstående alternativen

---

14. Antag att du vill implementera fakultet i Python, d.v.s. en funktion f som för ett givet heltal n returnerar $n(n-1)(n-2)\cdots 1$ som en rekursiv funktion. (T.ex. ska $f(3)$ returnera 6 eftersom $3 \cdot 2 \cdot 1 = 6$.) Vad måste då stå på den femte raden i koden

```
def f(n):
    if n==1:
        return 1
    else:
```

(Du kan utgå från att argumentet, `n`, är ett positivt heltal.) (1p)

(a) `return f(f(n)-1)`

(b) `return f(f(n-1))`

(c) `return n*(n-1)*(n-2)**1`

(d) `return f(n-1)`

(e) `return n*f(n-1)`

(f) `return (n-1)*f(n)`

(g) Inget av de ovanstående alternativen

Skrivfrågor

15. Vad är den 3-bokstaviga förkortningen för de datorkomponent som man brukar kalla "datorns närminne"? (1p)

---

16. Hur kan man skriva en funktion `share` som tar en tupel bestående av minst 2 booleska värden som argument och skriver ut, med 4 decimalers noggrannhet, hur stor andel av dem som är `True`? (2p)

---

17. Vad svarar Python om man exekverar koden

```
d = {"a":4, "b":7, "2":"c"}
for k,v in d.items():
    if v>5:
        print(k + ": " + str(v), sep=" *** ")
    else:
        pass
```

(Om du tror att det blir "Error" måste du specificera vilken sorts "Error".) (2p)

---

18. Låt `a` och `b` vara två mängder. Hur kan man i Python bilda den mängd `c` som innehåller alla element i `a` och alla element i `b`? (1p)

---

19. Vad måste stå på första (tomma) raden i följande kod

```
_____________________________
    f.write(input("Skriv en enradsdikt! "))
```

så att den text man matar in på förmaningen `Skriv en enradsdikt:` hamnar i filen `dikt.txt`? (2p)

---

20. Antag att modulen `random` är importerad. Skriv ett program som efterfrågar ett heltal `n` större än 3 och som output i terminalen ger ett lösenord slumpmässigt sammansatt av `n` bokstäver `a-z`. (3p)

---

21. En variabel `d` av typen dictionary innehåller par av strängar. Hur kan man skriva tre rader så att, för alla key-value-par i `d`, om key-strängen har ändelsen `ab` så ska ändelsen `cd` läggas till i value-strängen? (2p)

---

22. Skriv funktionen `notdiv` som ska ta ett heltal större än 2 som argument, `n`, och returnera hur många positiva heltatal mellan `2` och `n-1` som `n` inte är jämnt delbart med. (3p)

Python 3 Cheat Sheet

Latest version on :
https://perso.limsi.fr/pointal/python:memento

Base Types

integer, float, boolean, string, bytes

```
int 783  0 -192  0b010 0o642 0xF3
          zero   binary  octal  hexa
float 9.23 0.0  -1.7e-6
                    ×10^-6
bool True  False
str "One\nTwo"        Multiline string:
     escaped new line  """X\tY\tZ
      'I\'m'           1\t2\t3"""
      escaped '         escaped tab
bytes b"toto\xfe\775"
          hexadecimal  octal
```

immutables

Container Types

- **ordered sequences**, fast index access, repeatable values

```
list [1,5,9]   ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)   11,"y",7.4    ("mot",)  ()
```

Non modifiable values (immutables) — *expression with only comas →* `tuple`

```
str bytes   (ordered sequences of chars / bytes)   ""  b""
```

- **key containers**, no *a priori* order, fast key access, each key is unique

```
dictionary  dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection  set {"key1","key2"}   {1,9,3,0}   set()
```

keys=hashable values (base types, immutables…) `frozenset` *immutable set* *empty*

Identifiers

for variables, functions, modules, classes… names

`a…zA…Z_` followed by `a…zA…Z_0…9`

- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination

☺ `a toto x7 y_max BigOne`
☹ ~~`for`~~ ~~`and`~~ ~~`for`~~

Variables assignment

`=`

assignment ⇔ **binding** of a *name* with a *value*

1) evaluation of right side expression value
2) assignment in order with left side names

```
x=1.2+8+sin(y)
a=b=c=0          assignment to same value
y,z,r=9.2,-7.6,0 multiple assignments
a,b=b,a          values swap
a,*b=seq  }  unpacking of sequence in
*a,b=seq  }  item and list
x+=3   increment ⇔ x=x+3      and *= /= %= …
x-=2   decrement ⇔ x=x-2
x=None « undefined » constant value
del x  remove name x
```

Conversions

`type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63        can specify integer number base in 2nd parameter
int(15.56) → 15          truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6      rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→"…"   representation string of x for display (cf. formatting on the back)
chr(64)→'@'  ord('@')→64   code ↔ char
repr(x)→"…"  literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
   ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
   "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
   "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
   [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Sequence Containers Indexing

for lists, tuples, strings, bytes…

```
negative index  -5  -4  -3  -2  -1
positive index   0   1   2   3   4
lst=           [10, 20, 30, 40, 50]
positive slice  0   1   2   3   4   5
negative slice -5  -4  -3  -2  -1
```

Items count
`len(lst)→5`

index from 0
(here from 0 to 4)

Individual access to **items** via `lst[`*index*`]`

```
lst[0]→10   ⇒ first one      lst[1]→20
lst[-1]→50  ⇒ last one       lst[-2]→40
```

On mutable sequences (`list`), remove with `del lst[3]` *and modify with assignment* `lst[4]=25`

Access to **sub-sequences** via `lst[`*start slice*`:`*end slice*`:`*step*`]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]     lst[::-2]→[50,30,10]         lst[-3:-1]→[30,40]  lst[3:]→[40,50]
lst[::2]→[10,30,50]      lst[:]→[10,20,30,40,50]  shallow copy of sequence
```

Missing slice indication → from start / up to end.
On mutable sequences (`list`), remove with `del lst[3:5]` *and modify with assignment* `lst[1:4]=[15,25]`

Boolean Logic

Comparisons : `<` `>` `<=` `>=` `==` `!=` (*boolean results*) — ≤ ≥ = ≠

`a and b` logical and — *both simultaneously*

`a or b` logical or — *one or other or both*

pitfall : `and` *and* `or` *return* *value* *of* `a` *or* `b` *(under shortcut evaluation).*
⇒ *ensure that* `a` *and* `b` *are booleans.*

`not a` logical not

`True` `False` — True and False constants

Statements Blocks

```
parent statement:
    statement block 1…
    ⋮
parent statement:
    statement block2…
    ⋮
next statement after block 1
```

indentation !

configure editor to insert 4 spaces in place of an indentation tab.

Modules/Names Imports

module `truc` ⇔ *file* `truc.py`

```
from monmod import nom1,nom2 as fct
```
→*direct access to names, renaming with* `as`

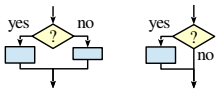
```
import monmod
```
→*access via* `monmod.nom1` …

modules and packages searched in *python path* (cf `sys.path`)

Conditional Statement

statement block executed only if a condition is true

```
if logical condition:
    statements block
```



Can go with several *elif*, *elif*... and only one final *else*. Only the block of first true condition is executed.

```
if age<=18:
    state="Kid"
elif age>65:
    state="Retired"
else:
    state="Active"
```

with a var `x`*:*
`if bool(x)==True:` ⇔ `if x:`
`if bool(x)==False:` ⇔ `if not x:`

Maths

floating numbers… approximated values *angles in radians*

Operators: `+` `-` `*` `/` `//` `%` `**`
Priority `(…)` — × ÷ ↑ ↑ a^b — integer ÷ ÷ remainder

`@` → matrix × *python3.5+*`numpy`

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

usual order of operations

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

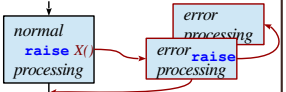
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

Exceptions on Errors

Signaling an error:
`raise` *ExcClass(…)*

Errors processing:

```
try:
    normal procesising block
except Exception as e:
    error processing block
```



`finally` *block for final processing in all cases.*

Conditional Loop Statement

*statements block executed **as long as** condition is true*

```
while logical condition:
    statements block
```


```
s = 0          } initializations before the loop
i = 1          }
while i <= 100:   # condition with a least one variable value (here i)
    s = s + i**2
    i = i + 1     # make condition variable change !
print("sum:",s)
```

Algo: $s = \sum_{i=1}^{i=100} i^2$

beware of infinite loops!

Loop Control

`break` *immediate exit*

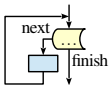
`continue` *next iteration*

`else` *block for **normal** loop exit.*

Iterative Loop Statement

*statements block executed **for each** item of a container or iterator*

```
for var in sequence:
    statements block
```



Go over sequence's **values**

```
s = "Some text"   } initializations before the loop
cnt = 0           }
for c in s:   # loop variable, assignment managed by for statement
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

Algo: count number of `e` *in the string.*

loop on dict/set ⇔ loop on keys sequences
use *slices* to loop on a subset of a sequence

Go over sequence's **index**

- modify item at index
- access items around index (before / after)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Algo: limit values greater than 15, memorizing of lost values.

Go simultaneously over sequence's **index** and **values**:

```
for idx,val in enumerate(lst):
```

good habit : don't modify loop variable

Display

```
print("v=",3,"cm :",x,",",y+4)
```

items to display : literal values, variables, expressions

`print` options:

- `sep=" "` items separator, default space
- `end="\n"` end of print, default new line
- `file=sys.stdout` print to file, default standard output

Input

```
s = input("Instructions:")
```

`input` always returns a **string**, convert it to required type (cf. boxed *Conversions* on the other side).

Generic Operations on Containers

`len(c)` → items count
`min(c)` `max(c)` `sum(c)`
`sorted(c)` → `list` sorted *copy*
`val in c` → boolean, membership operator `in` (absence `not in`)
`enumerate(c)` → *iterator* on (index, value)
`zip(c1,c2…)` → *iterator* on tuples containing c_i items at same index
`all(c)` → `True` if **all** `c` items evaluated to true, else `False`
`any(c)` → `True` if **at least one** item of `c` evaluated true, else `False`

*Note: For dictionaries and sets, these operations use **keys**.*

*Specific to **ordered sequences containers** (lists, tuples, strings, bytes…)*

`reversed(c)` → *inversed iterator* `c*5` → duplicate `c+c2` → concatenate
`c.index(val)` → *position* `c.count(val)` → *events count*

```
import copy
copy.copy(c) → shallow copy of container
copy.deepcopy(c) → deep copy of container
```

Integer Sequences

`range([start,] end [,step])`

start default 0, *end* not included in sequence, *step* signed, default 1

`range(5)` → 0 1 2 3 4 `range(2,12,3)` → 2 5 8 11
`range(3,8)` → 3 4 5 6 7 `range(20,5,-5)` → 20 15 10
`range(len(seq))` → *sequence of index of values in seq*

range provides an immutable sequence of int constructed as needed

Operations on Lists

modify original list

`lst.append(val)`	add item at end
`lst.extend(seq)`	add sequence of items at end
`lst.insert(idx,val)`	insert item at index
`lst.remove(val)`	remove first item with value *val*
`lst.pop([idx])` → *value*	remove & return item at index *idx* (default last)
`lst.sort()` `lst.reverse()`	sort / reverse liste *in place*

Operations on Dictionaries

`d[key]=value` `d.clear()`
`d[key]` → *value* `del d[key]`
`d.update(d2)` update/add associations
`d.keys()` `d.values()` `d.items()` → *iterable views on keys/values/associations*
`d.pop(key[,default])` → *value*
`d.popitem()` → *(key,value)*
`d.get(key[,default])` → *value*
`d.setdefault(key[,default])` → *value*

Operations on Sets

Operators:
`|` → union (vertical bar char)
`&` → intersection
`-` `^` → difference/symmetric diff.
`<` `<=` `>` `>=` → inclusion relations
Operators also exist as methods.

`s.update(s2)` `s.copy()`
`s.add(key)` `s.remove(key)`
`s.discard(key)` `s.clear()`
`s.pop()`

Function Definition

function name (identifier)
named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

`fct`

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`
**args variable positional arguments (→`tuple`), default values,*
***kwargs variable named arguments (→`dict`)*

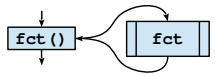
Function Call

```
r = fct(3,i+2,2*i)
```

storage/use of returned value *one argument per parameter*

this is the use of function name *with parentheses* which does the call

*Advanced: *sequence **dict*



Operations on Strings

`s.startswith(prefix[,start[,end]])` `s.endswith(suffix[,start[,end]])` `s.strip([chars])`
`s.count(sub[,start[,end]])` `s.partition(sep)` → *(before,sep,after)*
`s.index(sub[,start[,end]])` `s.find(sub[,start[,end]])`
`s.is…()` *tests on chars categories (ex. `s.isalpha()`)*
`s.upper()` `s.lower()` `s.title()` `s.swapcase()`
`s.casefold()` `s.capitalize()` `s.center([width,fill])`
`s.ljust([width,fill])` `s.rjust([width,fill])` `s.zfill([width])`
`s.encode(encoding)` `s.split([sep])` `s.join(seq)`

Formatting

formating directives — values to format

```
"modele{} {} {}".format(x,y,r) → str
"{selection:formatting!conversion}"
```

- **Selection** :
  `2`
  `nom`
  `0.nom`
  `4[key]`
  `0[2]`

Examples:
`"{:+2.3f}".format(45.72793)` → `'+45.728'`
`"{1:>10s}".format(8,"toto")` → `'      toto'`
`"{x!r}".format(x="I'm")` → `'"I\'m"'`

- **Formatting** :

fill char *alignment* *sign* *mini width*.*precision~maxwidth* *type*

`< > ^ =` `+ - space` `0` at start for filling with 0

integer: `b` binary, `c` char, `d` decimal (default), `o` octal, `x` or `X` hexa…
float: `e` or `E` exponential, `f` or `F` fixed point, `g` or `G` appropriate (default),
string: `s` … `%` percent

- **Conversion** : `s` (readable text) or `r` (literal representation)

Files

storing data on disk, and reading it back

```
f = open("file.txt","w",encoding="utf8")
```

file **variable** for operations
name of file on disk (+path…)
opening **mode**
- `'r'` read
- `'w'` write
- `'a'` append
- …`'+'` `'x'` `'b'` `'t'`

encoding of chars for *text files*: `utf8` `ascii` `latin1` …

cf. modules `os`, `os.path` and `pathlib`

writing

```
f.write("coucou")
f.writelines(list of lines)
```

reading

read empty string if end of file

```
f.read([n])          → next chars
                       if n not specified, read up to end !
f.readlines([n])     → list of next lines
f.readline()         → next line
```

*text mode `t` by default (read/write `str`), possible binary mode `b` (read/write `bytes`). **Convert from/to required type** !*

`f.close()` dont forget to **close the file** after use !

`f.flush()` write cache `f.truncate([size])` resize

reading/writing progress sequentially in the file, modifiable with:
`f.tell()` → *position* `f.seek(position[,origin])`

Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(…) as f:
    for line in f :
        # processing of line
```