Tentamen i Programmering DI2006

Datum: 2023-01-09
Tid: 15.00–19.00
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.
- Frågorna i Del 2 ska besvaras på vanligt separat rutat papper.
- Då programkod anges som svar ska den vara i så körbart skick som möjligt.
- Del 1 består av 23 frågor och här kan man maximalt få 30 poäng.
- Del 2 består av 2 frågor och här kan man maximalt få 15 poäng.
- För betyg 3 krävs minst 15 poäng på Del 1. (Del 2 behöver inte alls göras för att få detta betyg.)
- För betyg 4 krävs minst 15 poäng på Del 1 och 7 poäng på Del 2.
- För betyg 5 krävs minst 15 poäng på Del 1 och 11 poäng på Del 2.

LYCKA TILL!

Del 1

Flervalsfrågor

1. Vad kallas det talsystem som bara använder siffrorna 0 och 1 för att bilda alla andra tal? (1p)
   (a) Det tvådimensionella talsystemet
   (b) Det duala talsystemet
   (c) Det hexadecimala talsystemet
   (d) Det binära talsystemet
   (e) Det oktala talsystemet
   (f) Det logiska talsystemet
   (g) Inget av de ovanstående alternativen

---

2. Arbetsminnet brukar man ibland kalla den del av datorn som egentligen heter (1p)
   (a) LCD
   (b) AMD
   (c) Moderkortet
   (d) CPU
   (e) HDD
   (f) RAM
   (g) Inget av de ovanstående alternativen

---

3. I varje rekursiv funktion måste (1p)
   (a) programspråket vara imperativt
   (b) funktionen anropa sig själv
   (c) första raderna utgöras av en konstruktor
   (d) alla variabler vara heltal
   (e) minst en `if`-sats förekomma
   (f) det börja och sluta med ett `return`-kommando
   (g) Inget av de ovanstående alternativen

---

4. Betrakta Pythonkoden:

```
varv = 1
for i in "abrafax":
    varv += 1
print(varv)
```

Vad händer när man exekverar den? (1p)

(a) `1` skrivs ut

(b) `7` skrivs ut

(c) `8` skrivs ut

(d) `abrafax` skrivs ut

(e) Datorn ”hänger sig” (dvs det blir en oändlig loop)

(f) Ingenting

(g) Inget av de ovanstående alternativen

---

5. I en variabel av typen `dictionary` är elementen par av variabler. Första halvan av varje sådant par kallas: (1p)

(a) item

(b) key

(c) value

(d) element

(e) entry

(f) one

(g) Inget av de ovanstående alternativen

---

6. Vad ska man skriva på den tomma raden i koden:

```
lista1 = [1,4,2,8]
lista2 = ____________________
for i in range(len(lista1)):
     lista1[i] = 9-lista2[i]
print(lista2[3])
```

för att talet 8 ska skrivas då man kör koden? (1p)

(a) `lista1[0:-1]`

(b) `lista1[:]`

(c) `lista1`

(d) `[1,1,1,1]`

(e) `list(range(4))`

(f) `[4]*4`

(g) Inget av de ovanstående alternativen

---

7. Om man exekverar koden

```
t = (2,3,5,7,11)
for i in range(len(t)):
    t[i] = t[(i+1)%5]
    print(t[i],end="")
```

svarar Python: (1p)

(a) `235711`

(b) `357112`

(c) `112357`

(d) `IndexError`

(e) `ZeroDivisionError`

(f) `TypeError`

(g) Inget av de ovanstående alternativen

---

8. Vad ska stå på den tomma raden i koden:

```
f = open( ____________________ )
f.write("Gurka")
f.close()
```

så att ordet `Gurka` läggs till den text som redan finns i filen `Frukt.txt`? (1p)

(a) `"ta","Frukt.txt"`

(b) `"br","Gurka.txt"`

(c) `"Frukt.txt","ta"`

(d) `"Gurka.txt","br"`

(e) `"x","Frukt"`

(f) `"Frukt","xw"`

(g) Inget av de ovanstående alternativen

---

9. Antag att variabeln `language` har värdet `"Python"`. Vilket av följande kommandon ändrar alla dess tecken till gemener (dvs ”små bokstäver”)? (1p)

(a) `small(language)`

(b) `lower(language)`

(c) `language = language.lower()`

(d) `language = language.small()`

(e) `language.small()`

(f) `language.lower()`

(g) Inget av de ovanstående alternativen

---

10. Låt variablerna `name` och `length` vara definierade enligt

```
name = "Ian"
length = "179"
```

Om utskriften

```
I am Ian and I'm 1.8 meters tall.
```

skrivs ut, vilket av följande kommandon kan ha orsakat det: (1p)

(a) `print(name,length)`

(b) `print("I am "+name+", and I'm "+f'str(length)/100:.1f'+" meters tall.")`

(c) `print("I am ",name,", and I'm ",f'int(length)*100:.1f',"meters tall.")`

(d) `print("I am "+name+", and I'm "+str(f'int(length)//100:.1f')+" meters tall.")`

(e) `print("I am "+name+", and I'm "+f'{int(length)/100:.1f}'+" meters tall.")`

(f) `print("I am ",name,", and I'm ",str(f'int(length)%100:.1f'),"meters tall.")`

(g) Inget av de ovanstående alternativen

---

11. Låt `abc = 'abcdefghijklmnopqrstuvwxyzåäö'`. Vilket av följande alternativ ger ordet `tenta`? (1p)

(a) `abc[20]+abc[5]+abc[14]+abc[20]+abc[1]`

(b) `abc[5]+abc[20]+abc[14]+abc[5]+abc[20]`

(c) `(abc[0]+abc[19]+abc[13]+abc[4]+abc[19]).reverse()`

(d) `abc[-9]+abc[-24]+abc[-15]+abc[-9]+abc[-28]`

(e) `abc[19]+abc[-15]+abc[-6]+abc[19]+abc[0]`

(f) `abc[-10]+abc[4]+abc[13]+abc[-10]+abc[-29]`

(g) Inget av de ovanstående alternativen

---

12. Koden

```
days = ['Mon','Tue']
days.append(['Fri','Sat'])
```

ger variabeln `days` värdet: (1p)

(a) `['Mon','Tue',['Fri','Sat']]`

(b) `['Mon','Tue','Fri','Sat']`

(c) `[['Fri','Sat'],'Mon','Tue']`

(d) `['Fri','Sat','Mon','Tue']`

(e) `['Mon','Fri','Tue','Sat']`

(f) `[['Sat'],'Mon',[Fri],'Tue']`

(g) Inget av de ovanstående alternativen

---

13. Låt `M` vara en variabel med värdet `[[1,3,5],[2,4,6],[7,8,9]]`. Vilket av följande alternativ får värdet `-1`? (1p)

    (a) `M[1][2]//M[0][1]`

    (b) `M[-M[0][1]][2]-M[2][0]`

    (c) `M[0][0]**M[-1][-1]`

    (d) `M[2][-1]/M[1][-2]`

    (e) `M[0][M[1][-3]]-M[-M[-3][1]][0]`

    (f) `M[-1][-2]%M[1][0]-M[-3][-M[-3][-2]]`

    (g) Inget av de ovanstående alternativen

---

14. Vilket av följande alternativ är en fördefinierad metod vid definition av klasser på samma sätt som metoden `__init__(...)`? (1p)

    (a) `__let__(...)`

    (b) `__int__(...)`

    (c) `__str__(...)`

    (d) `__open__(...)`

    (e) `__clean__(...)`

    (f) `__def__(...)`

    (g) Inget av de ovanstående alternativen

Skrivfrågor

15. Vad kallas det kretskort där mikroprocessorn (CPU:n), arbetsminnet (RAM-minnet) och flera andra komponenter sitter? (1p)

---

16. Vad kallas den datatyp som antar värdena `True` och `False`? (1p)

---

17. Hur skriver man i Python för att datorn ska fråga
`What is your name?`
och då man svarat genom att skriva sitt namn <n> i sin tur svarar
`Hello <n>! My name is HAL 9000.` (2p)

---

18. Ange den kod som ska stå på den streckade raden så att den ger utskriften (2p)

```
Mia      7
Gustav   3
```

då man exekverar:

```
temp={"Mia":6,"Gustav":2}
```

```
    print(x,"\t",y+1)
```

---

19. Vad svarar Python om man exekverar koden (2p)

```
a = (2,3)
try:
    a.append(1/(a[0]-2))
    print(a[2])
except ZeroDivisionError:
    print(a[0])
```

---

20. Låt `u` vara en lista med heltal eventuellt med dubletter (dvs element som förekommer *minst* två gånger). Hur kan man **på en rad** konstruera en lista `v` som innehåller alla heltal i `u`, inga andra heltal och inga dubletter. (2p)

---

21. I vilken ordningsföljd exekveras följande kod:

```
1  def ackermann(m,n):
2      if m==0:
3          return n+1
4      elif n==0:
5          return ackermann(m-1,n)
6      else:
7          return ackermann(m-1,ackermann(m,n-1))
8
9  ackermann(1,1)
```

Ange sekvensen av radnummer som svar. (2p)

---

22. Hur kan man skriva kod som efterfrågar ett tal, och om talet är jämnt svarar `Jämnt` och om det är udda svarar `Udda`? (2p)

---

23. Hur kan man tilldela variabeln `t` värdet av ett virtuellt tärningskast, dvs ett slumpmässigt tal mellan 1 och 6? (2p)

Del 2

Programmeringsuppgifter

24. *Bala taklänges*
Skriv ett program som tar en text som indata och vänder på ordningsföljden av tecknen i skriver ut denna text. Dock ska en initial versal (dvs stor bokstav) i ett ord göra att motsvarande baklängesord får avslutande versal då det läses baklänges. T.ex. ska texten
`Det var en gång en gosse som hette Hans och en flicka som hette Greta`
rendera baklängestexten
`Aterg etteh mos akcilf ne hco Snah etteh mos essog ne gnåg ne rav Ted`
Lös denna uppgift
   (a) med vilka med vilka funktioner du vill. (3p)
   (b) genom att använda minst en rekursiv funktion. (3p)

25. *Innehållsförteckning*
Skriv ett program som består av en funktion som läser in en från filen `text.txt`. Texten är en uppsats som förutom vanligt text innehåller kommandona `\chapter{...}` och `\section{...}`. Varje kapitel- och sektionskommando kommer på egen rad. Då dessa kommandon förekommer ska det bildas kapitelnamn respektive sektionsnamn i uppsatsen. Din uppgift är dock bara att göra en innehållsförteckning för denna uppsats. Därmed måste du ha koll på vilka sidor olika rubriker kommer. Överst i outputen ska rubriken `Table of contents:` stå. För att avgöra sidindelningen ska uppsatsen ha 100 ord ur vanlig text per sida (dvs ord ur kapitel- och sektionsrubrikerna ska ej räknas). Dessutom ska kapitel och sektioner numreras korrekt. Detta innebär att kapitlena ska numreras `1`, `2`, `3` osv i tur och ordning medan sektionsnumren anges med två tal $X.Y$ där X är numret på det kapitel där sektionen förekommer och Y är sektionsnumret som börjar om på `1` för varje nytt kapitel. Utrymmet mellan kapitel- respektive sektionsposterna och deras sidnummer ska vara fyllt av en punktad linje så att varje rad utgörs av exakt n tecken där talet n ska ges som input från kommandoraden. Sedan ska innehållsförteckningen där alla kapitel- och sektionstitlar och respektive sidnummer är angivna skrivas till filen `innehall.txt`.
   (a) Skriv den funktion, `distribution`, som läser in texten från filen `text.txt` och beräknar hur sidindelningen blir. Den ska också ha koll på när det kommer kapitel-, respektive sektionskommandon på dessa sidor. Funktionen ska returnera en lista av tupler där varje tupel är en kvadrupel. Första elementet i varje kvadrupel är antingen tecknet `"K"` (som i kapitel) eller tecknet `"S"` (som i sektion). Andra elementet är kapitel- respektive sektionsnumret. Tredje elementet i kvadrupeln är namnet på kapitlet respektive sektionen (dvs det som stod inom krullparenteserna, `{...}`, i texten). Fjärde elementet är sidnumret där detta kapitel respektive denna sektion finns. (5p)
   (b) Skriv sedan den funktion, `contents`, som tar listan med kvadrupeltupler som den får från funktionen `distribution` och genererar själva innehållsförteckningen. Denna ska innehålla alla uppsatsens kapitel och sektioner korrekt numrerade och namngivna och med rätt sidnummer enligt instruktionerna. Funktionen ska vara void-returning men skriva innehållsförteckningen till filen `innehall.txt`. (4p)

T.ex. ska filen `text.txt` enligt:

```
\chapter{Introduction}
Lorem ipsum dolor sit amet, consectetur adipiscing elit...
<90 ord>
... Donec vestibulum justo sit amet iaculis finibus.
\section{Background}
Mauris ipsum nulla, dictum vel eleifend eget, ullamcorper...
<356 ord>
... eget facilisis eu, semper nec mi.
\section{Research problem}
Donec pulvinar maximus tortor, id euismod felis sodales...
<292 ord>
... dignissim fermentum felis. Vivamus eu felis odio.
\chapter{Results}
Donec pulvinar maximus tortor, id euismod felis...
<184 ord>
... posuere. Curabitur eu ullamcorper ante.
\section{Empirical findings}
Nulla quam justo, elementum eu ligula pretium, feugiat...
<275 ord>
... facilisis eu, semper nec mi.
\section{Simulations}
Donec pulvinar maximus tortor, id euismod felis...
<308 ord>
... metus luctus viverra ac sit amet augue.
\chapter{Bibliography}
```

då man exekverar `contents(distribution(50))` resultera i innehållsförteckningen:

```
Table of contents:
1  Introduction ..............................  1
   1.1 Background ............................  1
   1.2 Research problem ......................  5
2  Results ...................................  8
   2.1 Empirical findings ..................... 10
   2.2 Simulations ........................... 12
3  Bibliography ............................... 16
```

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`

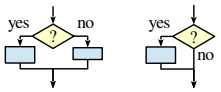
```
import monmod
```
→*access via* `monmod.nom1` …

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
```

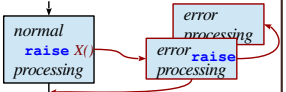
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

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*

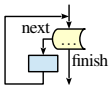
`continue` *next iteration*

`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`)*

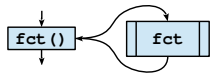
Function Call

```
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
```