Tentamen i Programmering DI2006

Datum: 2023-05-31
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 21 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 innebär begreppet RAM i datorsammanhang? Det är en slags (1p)
   (a) mikroprocessor
   (b) grafikkort
   (c) arbetsminne
   (d) hårddisk
   (e) procedur för problemlösning
   (f) bildskärm
   (g) Inget av de ovanstående alternativen

---

2. Hur anger man en kommentar i Python? Genom att (1p)
   (a) börja raden med tecknet #
   (b) börja raden med teckent $
   (c) börja raden med tecknet ;
   (d) börja raden med tecknet %
   (e) omgärda raden med tecknen (∗ respektive ∗)
   (f) omgärda raden med tecknen <!-- respektive -->
   (g) Inget av de ovanstående alternativen

---

3. Vilket av följande alternativ är ett lågnivåspråk? (1p)
   (a) Java
   (b) Assembly
   (c) Python
   (d) C
   (e) Fortran
   (f) PHP
   (g) Inget av de ovanstående alternativen

---

4. Vad kallas alltid den obligatoriska metoden vid defintionen av en klass? (1p)
   (a) `__str__`
   (b) `__start__`
   (c) `__let__`
   (d) `__func__`
   (e) `__init__`
   (f) `__return__`
   (g) Inget av de ovanstående alternativen

---

5. Gollum har skrivit koden

```
a = (1)
for i in range(100):
  if (i+10)//(1+3*i%10)<5:
    a.append(i)
print(a)
```

men får meddelandet `AttributeError` då han exekverar den. Varför? (1p)

(a) Gränsen `100` är för stor för metoden `range`

(b) Indexet `i` går utanför gränserna för objektet `a`

(c) Evalueringen av `(i+10)//(1+3*i%10)` ger division med 0

(d) Objektet `a` har inte attrubutet `append`

(e) Felaktig indentering av raderna 3 och 4

(f) Man kan inte tilldela ett objekt värdet `(1)`

(g) Inget av de ovanstående alternativen

---

6. Vad svarar Python då man exekverar koden (1p)

```
a = True
b = not a
print((a and b) or (a not in b))
```

(a) `0`

(b) `1`

(c) `True`

(d) `False`

(e) `TypeError`

(f) `bool`

(g) Inget av de ovanstående alternativen

---

7. Vad svarar Python då man exekverar koden (1p)

```
def f(a,b):
  if ((a and b) or (not a and b)):
    return(a**b)
  else:
    return(a or b)

f(False,True)
```

(a) `0`

(b) `1`

(c) `True`

(d) `False`

(e) `TypeError`

(f) `bool`

(g) Inget av de ovanstående alternativen

---

8. Datatypen `dict` (dictionary) är sekventiell med element bestående av par av värden. Vad kallas första halvan i varje sådant par? (1p)
   (a) `item`
   (b) `value`
   (c) `token`
   (d) `index`
   (e) `key`
   (f) `init`
   (g) Inget av de ovanstående alternativen

---

9. Vad kallas den typ av programmering som handlar om att skapa funktioner som är separerade från de data de hanterar (1p)
   (a) imperativ
   (b) relationsbaserad
   (c) objektorienterad
   (d) maskinkod
   (e) lambdakalkyl
   (f) pseudokod
   (g) Inget av de ovanstående alternativen

---

10. Vad kallas det när man definierar en funktion genom att någon eller några av dess värden beräknas med hjälp av anrop till funktionen själv (1p)
   (a) loop
   (b) rekursion
   (c) rekurrens
   (d) självreferens
   (e) induktion
   (f) feedback
   (g) Inget av de ovanstående alternativen

---

11. Vilken numerisk operator har innebörden av heltalsdivision (d.v.s. den ger heltalsdelen av kvoten vid division av två tal)? (1p)
   (a) `*`
   (b) `**`
   (c) `/`
   (d) `//`
   (e) `%/%`
   (f) `div`
   (g) Inget av de ovanstående alternativen

---

12. Vilken operator har innebörden av konkatenering (d.v.s. sammanfogning) av två listor? (1p)

    (a) `*`

    (b) `/`

    (c) `+`

    (d) `-`

    (e) `conc`

    (f) `^`

    (g) Inget av de ovanstående alternativen

---

13. Hur kan man skriva för att värdet av variabeln `a`, som genereras vid exekveringen av ett Pythonprogram, ska skrivas till filen `output.txt`? (1p)

    (a) `write(a,output.txt,"w")`

    (b) `print(file=output.txt,a)`

    (c) `output(output.txt,"a")`

    (d) `write(a,"output.txt")`

    (e) `print("output.txt",a)`

    (f) `output(a,"output.txt","w")`

    (g) Inget av de ovanstående alternativen

---

14. Vilken inbyggd funktion i Python kan ovandla en variabel av typen `int` till en variabel av typen `str`? (1p)

    (a) `int2str`

    (b) `int-str`

    (c) `int`

    (d) `str`

    (e) `char`

    (f) `float`

    (g) Inget av de ovanstående alternativen

Skrivfrågor

15. Listan `words` innehåller endast strängar. Hur kan man på en rad skriva Pythonkod så att `iwords` blir listan av de strängar i `words` som innehåller bokstaven `i`? (2p)

---

16. Komplettera koden

```
a = ["tomat","gurka","krocket"]
with ...
  ...
```

på de två ställena det står `...` så att elementen ur listan `a` skrivs till den ännu icke existerande filen `diverse.txt`. (2p)

---

17. Komplettera koden (2p)

```
def f(x):
  p = 1
  ...
    p *= int(n)
  return(p)
```

så att anropet `f(x)` gör att produkten av de siffror som utgör talet `x` beräknas. T.ex. ska funktionen vid anropet `f(1234)` returnera talet `24` eftersom $1 \cdot 2 \cdot 3 \cdot 4 = 24$.

---

18. En telefonkatalog är sparad som en dictionary kallad `tfnnr`. Första halvan i varje element i `tfnnr` är ett namn i formen av en sträng och andra halvan av samma element är motsvarande telefonnummer i formen av ett heltal[1]. Hur ska man skriva om man vill veta telefonnumret till en person som heter Alfons Åberg? (2p)

---

19. Gustvard vill skriva en funktion som är en digital roulette. För varje gång som funktionen `roulette` exekveras så ska den returnera en tupel av ett slumpmässigt heltal mellan 1 och 36 och en färg i form av en sträng. Om heltalet är något av talen i listan
`farg = [1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36]`
så ska färgen vara `"Rött` och annars ska den vara `Svart`. Antag att listan `farg` och raden `import random` är givna. Hur kan Gustvard då skriva sin funktion `roulette` på endast 3 rader? (3p)

---

20. Vad skrivs ut om man exekverar programmet (2p)

```
def f(a):
  if a>20:
    return(int(a**0.5))
  else:
    return(a*f(7*a+1))

f(1)
```

---

[1] Man får anta att alla personer i katalogen har olika namn och att inget telefonnummer börjar på siffran 0 men bortse från sådana teknikaliteter.

21. Skriv ett växlingsprogram `forex.py` som efterfrågar `Kronor?` och `Ören?` och sedan svarar med motsvarande `Detta blir ... Euro och ... Cent` enligt växelkursen 1 euro = 11.51 kronor. (3p)

Del 2

Programmeringsuppgifter

22. I ett bibliotek finns alla böckernas titlar i en lista. Med hjälp av denna lista har man sedan konstruerat en dictionary där varje låntagare finns angiven intill respektive boktitel om boken är utlånad. Med följande kod ska en presumtiv låntagare kunna fråga om boken finns inne, få reda på vilka böcker som finns lånade av en viss låntagare och få en fullständig listning av alla låntagare och deras lån. Biblioteket har ett *låneregister* `rec` är består av en dictionary bestående av samtliga böckers titlar som `keys` samt namnet på den låntagare som har lånat boken för tillfället som `values`:
`rec = {'Rosens namn':'Bertil', 'Da Vinci koden':'Bertil', 'Do androids dream of electric sheep?':'Åsa', 'Neuromancer':'Åsa', 'Digital fortress':'Edwin', 'Kodboken':'Edwin', 'The hydrogen sonata':'Åsa', 'Foucaults pendel':'Bertil', 'biblen':'', 'Kvartetten som sprängdes':'', 'Röda rummet':'', 'Gösta Berlings saga':'', 'Childhoods end':'','Wasp factory':''}`
Observera att om en bok inte är utlånad är motsvarande `value` en tom sträng. Skriv nu programmet `biblio` som administrerar låneregistret. Då programmet startas ska användaren göra ett val: `F`, `A` eller `N` där (2p)

(a) `F` innebär att vill veta om en viss bok finns inne. Man ska då ange titeln på boken och programmet svarar: `Ulånad` om den redan är lånad av en annan person men om inte: `Ja, den finns inne. Vill du låna den?` följt av `Ok` om användaren svarar `Nej` men om användaren svarar `Ja` ska frågan följas av `Vad heter du?` och en uppdatering av låneregistret. (3p)

(b) `A` ska ge en listning av alla bibliotekets boktitlar oavsett om de är utlånade eller ej. (2p)

(c) `N` gör att man får uppmaningen `Ange ditt namn:` och då man skrivit sitt namn får beskedet `Du har för tillfället lånat:` följt av en listning av de boktitlar som finns i registret under det namnet. (3p)

---

23. *Sudoku*
Detta är en slags sifferpussel som började publiceras i svenska dagstidningar 2 juni 2005. Sudoko består av ett 9×9 rutnät. Varje ruta är platsen för ett siffra `1, 2, 3, ... , 9`. Reglerna för hur rutnätet får fyllas i är:

- Var och en av de 9 raderna ska innehålla varja siffra bland `1, 2, 3, ... , 9` en och endast en gång.
- Var och en av de 9 kolumnerna ska innehålla varje siffra bland `1, 2, 3, ... , 9` en och endast en gång.
- Rutnätets 81 rutor kan delas in i 9 stycken lika stora kvadrater var och en innehållande 9 rutor. Varje sådan kvadrat med 9 rutor ska innehålla varje siffra bland `1, 2, 3, ... , 9` en och endast en gång.

Du ska i denna uppgift åstadkomma en sudokugenerator, dvs ett program som ger en slumpmässig sudokulösning med alla rutorna ifyllda varje gång det körs.

(a) Skriv funktionen `one_line` som inte tar något argument men som ger en slumpmässig rad av siffrorna `1, 2, 3, ... , 9` i formen av en lista. (1p)

(b) Skriv funktionen `transpose` som tar en lista, `line`, och ett heltal, `k`, som argu-

ment. Den ska sedan förskjuta elementen i listan `k` steg, dvs om `line` innehåller elementen $x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_k$ så blir den `n` steg förskjutna listan listan med elementen $x_{n+1}, \ldots, x_k, x_1, x_2, \ldots, x_n$ (dvs de `n` första elementen ur `line` har lagts sist istället). Den förskjutna listan ska returneras av funktionen.(2p)

(c) Skriv funktionen `full_sudoku` som inte tar något argument men producerar en fullskalig sudoku på följande vis. Först ska den första raden genereras med hjälp av `one_line`. Sedan ska den andra raden bli likadan som den första men med siffrorna förskjutna 3 steg framåt. Sedan, i den tredje raden, ska siffrorna vara *ytterligare* 3 steg förskjutna. I fjärde, femte och sjätte raden ska siffrorna vara som i första andra och tredje men ytterligare ett steg förskjutna. I sjunde, åttonde och nionde raderna ska siffrorna vara ytterligare 1 steg förskjutna jämfört med rad fyra, fem och sex. Funktionen ska returnera en lista av listor där första inre listan är rad 1, andra inre listan är rad 2, osv. Den ska också skriva ut hela konfigurationen av alla siffrorna. Konfigurationen med alla blocken slumpade på detta vis ska returneras av funktionen. (3p)

(d) Skriv funktionen `block_sudoku` som inte heller tar något argument men som ger en något mer slumpmässig sudoku på följande vis. Först genereras en sudokukonfiguration med hjälp av funktionen `full_sudoku`. I denna konfiguration, kalla rad 1, 2 och 3 tillsammans för *block 1*, rad 4, 5 och 6 för *block 2* och rad 7, 8 och 9 för *block 3*. Dessa block ska komma i en slumpmässig ordning, dvs en ny 9×9-konfiguration ska konstrueras där blocken kan komma i vilken ordning som helst (1,2,3 eller 1,3,2 eller 2,1,3 eller 2,3,1 eller 3,1,2 eller 3,2,1). I den nya konfigurationen av alla siffrorna, kalla kolumn 1, 2 och 3 tillsammans för *block 4*, kolumn 4, 5 och 6 för *block 5* och kolumn 7, 8 och 9 för *block 6*. Dessa block ska också komma i slumpmässig ordning. (3p)

(e) Skriv slutligen `sudoku` som inte tar något argument men som producerar en helt slumpmässig sudokukonfiguration på följande vis. Först genereras en sudokukonfiguration med hjälp av funktionen `block_sudoku`. Sedan ska rad 1, 2 och 3 komma i slumpmässig ordning. Samma sak med rad 4, 5 och 6. Sedan samma sak med rad 7, 8 och 9. Vidare samma sak med kolumnerna: 1, 2 och 3 slumpmässigt. Kolumn 4, 5 och 6 slumpmässigt och kolumnerna 7, 8 och 9 slumpmässigt. Konfigurationen enligt denna slumpning av rader och kolumner ska returneras av funktionen. (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`

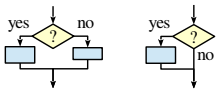
```
import monmod
```
→*access via* `monmod.nom1` …

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
```

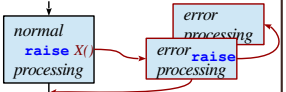
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

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*

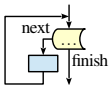
`continue` *next iteration*

`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`)*

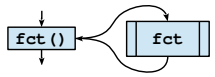
Function Call

```
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
```