Tentamen i Programmering DI2006

Datum: 2023-04-13
Tid: 9.00–13.00
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 till 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 24 frågor och här kan man maximalt få 30 poäng.
- Del 2 består av 2 frågor och här kan man maximalt få 20 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 9 poäng på Del 2.
- För betyg 5 krävs minst 15 poäng på Del 1 och 14 poäng på Del 2.

LYCKA TILL!

Del 1

Flervalsfrågor

1. Om en variabel tilldelas värdet 1.0 i Python så blir variabelns typ (1p)
   (a) `int`
   (b) `num`
   (c) `char`
   (d) `float`
   (e) `string`
   (f) `list`
   (g) Inget av de ovanstående alternativen

---

2. Operatorn som returnerar resten vid heltalsdivision mellan två tal är (1p)
   (a) `/`
   (b) `%`
   (c) `*`
   (d) `%/%`
   (e) `**`
   (f) `//`
   (g) Inget av de ovanstående alternativen

---

3. Antag att i Python man exekverar koden:

```
a,b,c = 1,1,1
while a*b < b+c:
    a += b
    b += c
    c += a
```

   Vilket värde får då variabeln `c`? (1p)
   (a) `1`
   (b) `3`
   (c) `5`
   (d) `7`
   (e) `9`
   (f) `11`
   (g) Inget av de ovanstående alternativen

---

4. En datatyp i Python som utgörs av en sekvens av `char` (dvs tecken) är (1p)
   (a) `var`
   (b) `char seq`
   (c) `dictionary`
   (d) `float`
   (e) `string`
   (f) `bool`
   (g) Inget av de ovanstående alternativen

---

5. Vad returneras i Python om man först definerar funktionen `f` enligt

```
def f(x):
    if x>10:
        return x//4
    elif x%3==0:
        return f(3*x+1)
    else:
        return f(2*x+1)
```

   och sedan skriver `f(4)`? (1p)
   (a) `0`
   (b) `1`
   (c) `4`
   (d) `7`
   (e) `9`
   (f) `ZeroDivisionError`
   (g) Inget av de ovanstående alternativen

---

6. Vad ska vid Pythonprogrammering stå på den tomma raden i koden

```
a = list(range(100))
b = []
for x in a:
    if x%7==0:
```

   _____

   för att `b` ska bli listan av de tal som är jämnt delbara med 7 mellan 0 och 100 (dvs 0, 7, 14, osv) och `a` ska bli listan av de andra talen mellan 0 och 100? (1p)
   (a) `a.append(b.pop(b.index(x)))`
   (b) `b.append(a.pop(a.index(x)))`
   (c) `a.pop(b.append(a.index(x)))`
   (d) `b.pop(a.append(b.index(x)))`
   (e) `a.index(b.pop(b.append(x)))`
   (f) `b.index(a.pop(a.append(x)))`
   (g) Inget av de ovanstående alternativen

---

7. Vid objektorienterad programmering vill man att metoderna ska vara "gömda", dvs ej tillgängliga för den som sedan använder klassen där de definieras. Hur kodas gömningen av metoderna? Genom att namnet på metoden (1p)

   (a) slutar på _ (en underscore)

   (b) slutar på __ (två underscore)

   (c) slutar på ___ (tre underscore)

   (d) börjar på _ (en underscore)

   (e) börjar på __ (två underscore)

   (f) börjar på ___ (tre underscore)

   (g) Inget av de ovanstående alternativen

---

8. Vilket av följande uttryck ger garanterat ett felmeddelande i Python? (1p)

   (a) `x > '7'`

   (b) `x = '7'`

   (c) `x and '7'`

   (d) `'7' or x`

   (e) `'7' = x`

   (f) `'7' < x`

   (g) Inget av de ovanstående alternativen

---

9. Vad kallas det kretskort i en dator som tolkar och bearbetar dess information till en videosignal som kan visas på en bildskärm? (1p)

   (a) Moderkort

   (b) RAM-minne

   (c) Ljudkort

   (d) Grafikkort

   (e) Mikroprocessor

   (f) Nätverkskort

   (g) Inget av de ovanstående alternativen

---

10. Python är *inte* ett språk som är (1p)

    (a) imperativt

    (b) objektorienterat

    (c) relationsbaserat

    (d) funktionellt

    (e) högnivå

    (f) programmeringsspråk

    (g) Inget av de ovanstående alternativen

---

11. Då koden

```
a = ''
while i in range(4):
    a[i+1] = a[i]+str(7//i)
    if a[3*i]==2:
        break
```

exekveras i Python genteraras felmeddelandet (1p)

(a) `TypeError`

(b) `KeyboardInterrupt`

(c) `ZeroDivisionError`

(d) `SyntaxError`

(e) `NameError`

(f) `IndexError`

(g) Inget av de ovanstående alternativen

---

12. En serie väldefinierade steg som genomförs för att utföra en viss uppgift kallas (1p)

(a) formellt uttryck

(b) logaritm

(c) algoritm

(d) syntax

(e) teorem

(f) paradigm

(g) Inget av de ovanstående alternativen

---

13. En lista i Python är begränsad av (1p)

(a) hakparenteser `[ ... ]`

(b) runda parenteser `( ... )`

(c) krullparenteser { `...` }

(d) citationstecken `" ... "`

(e) apostrofer `' ... '`

(f) pipes `| ... |`

(g) Inget av de ovanstående alternativen

---

14. En `if`-, `for`- eller `while`-sats i en annan `if`-, `for`- eller `while`-sats kallas för en (1p)

(a) nästlad sats

(b) itererad sats

(c) laggad sats

(d) aggregerad sats

(e) kaskadsats

(f) inkapslad sats

(g) Inget av de ovanstående alternativen

Skrivfrågor

15. Skriv Pythonkoden

```
if poang>100:
    vunnit = True
else:
    vunnit = False
if vunnit:
    print("Du vann! Grattis!!")
```

på två rader. (1p)

---

16. Vilket värde får variabeln `c` om man i Python kör följande kod? (1p)

```
a,b,c = 1,1,1
while a*b<b+c:
    a += b
    b += c
    c += a
```

---

17. Vad skrivs ut om man exekverar

```
a = 'tenta'
for i in range(len(a)-1):
    a = a[:i]+a[(2**i)%len(a)]+a[(i+1):]

print(a)
```

i Python? (2p)

---

18. Variabeln `ordlista` är en lista av strängar där varje sträng är ett ord. Hur kan man skriva en enradskod i Python som filtrerar ut alla ord som slutar på ändelsen *ning* ur listan `ordlista`? (1p)

---

19. I tupeln `stader` finns namnen på 10 städer (där varje stad är 8–12 bokstäver) som strängar. I tupeln `invanare` finns invånarantalen för de 10 städerna ur `stader`. Hur ska man programmera på tre rader i Python för att varje stadsnamn ska skrivas till filen `demografi.txt` följt av kolon med respektive invånarantal efter på en rad? (2p)

---

20. Antag att man har variabeln

```
dic = {'alfa':11, 'beta':22, 'gamma':33}
```

Hur kan man skriva 2 rader Pythonkod för att i outputfönstret lista dess keys i en kolumn och motsvarande values i en annan? (1p)

---

21. Antag att `digits` är en tom lista och att `n` är ett heltal mellan `1` och `1000`. Skriv högst 3 raders Pythonkod så att `digits` blir listan av de siffror som `n` består av.(2p)

---

22. Vad skrivs på skärmen när funktionen `mysko` körs? (2p)

```
def mysko(lista1, lista2):
    lista3 = []
    for i in lista1:
        if not i in lista2:
            lista3.append(i)
    for i in lista2:
        if not i in lista3:
            lista3.append(i)
    return lista3

print(mysko([1,2,3],[3,4,5]))
```

---

23. Skriv enradskoden i Python som efterfrågar `Hur många sekunder?` och svarar med så många hela minuter som detta räcker till. T.ex. om man anvivit `200` sekunder så ska svaret bli: `Det blir 3 hela minuter.` (2p)

---

24. Ett klot med radien r har volymen $V = \frac{4}{3}\pi r^3$. Skriv på två rader kod som efterfrågar en radie och svarar med volymen med 1 decimals noggrannhet för det klot som har den radien. Eventuella specialfunktioner måste importeras! (2p)

Del 2

Programmeringsuppgifter

25. *Dynamiskt rotationskrypto*
Följande krypteringssystem definieras:

- Den läsbara text som ska krypteras kallas *klartext* och den oläsbara text som den krypterats till kallas *kryptotext*.
- För krypteringen används klartextalfabetet `ABCDEFGHIJKLMNOPQRSTUVWXYZ` och kryptoalfabetet som är alla engelska alfabetets gemena (dvs små) bokstäver men inte i samma ordning som klartextalfabetet.
- Man anger vilken kryptoinställning det är fråga om med en tupel `(n1,n2)` av talen `n1` och `n2`.
- Den inbördes ordningen bland bokstäverna i kryptoalfabetet ska vara den alfabetiska ordningen men förskjuten `n1` steg.
- Det andra talet, `n2`, anger hur många steg kryptoalfabetet ska förskjutas ytterligare för kryptering av varje ny bokstav ur klartexten.

Exempel Antag att vi har klartexten `MEETING AT MIDNIGHT` och krypteringsnyckeln `(n1,n2) = (5,3)`. Med hjälp av `n1=5` får vi lathunden

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
f g h i j k l m n o p q r s t u v w x y z a b c d e
```

där kryptoalfabetet förskjutits 5 steg i för hållande till klartextalfabetet. Detta betyder att första bokstaven ur klartexten, `M`, ska bytas mot `r`. Därefter förskjuts kryptoalfabetet ytterligare `n2=3` steg så att vi för den andra bokstaven får

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
i j k l m n o p q r s t u v w x y z a b c d e f g h
```

varmed andra bokstaven, `E`, krypteras med `m`. I tredje steget fås

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
l m n o p q r s t u v w x y z a b c d e f g h i j k
```

(ytterligare 3 stegs förskjutning) och därmed byts `E` mot `p`. Totalt fås hela kryptotexten `rmphzhd ay uwuhfgmb`.

(a) Skriv funktionen `crypt` som tar krypteringsnyckeln `n1,n2` och klartexten i form av en sträng som argument (dvs indata) och returnerar värdet (dvs utdata): kryptotexten i form av en sträng. (3p)

(b) Skriv funktionen `decrypt` som gör tvärtom: tar krypteringsnyckeln `n1,n2` och kryptotexten i form av en sträng som argument (dvs indata) och returnerar värdet: klartexten i form av en sträng (dvs som utdata). (2p)

(c) Kryptoanalytikern är den som på något sätt har snappat upp kryptotexten men inte har tillgång till krypteringsnyckeln men ändå vill få reda på klartexten. Då kan det underlätta att ha tillgång till ett *orakel*. Detta är en person (eller ett program) som givet en bit information kan bidra till forceringen av kryptot. I detta fall vet oraklet de hemliga krypteringsnycklarna och med hjälp av dessa kan klartexter krypteras och kryptotexter dekrypteras. Om man skickar oraklet en klartext så krypteras denna med de hemliga krypteringsnycklarna och oraklet svarar med den motsvarande kryptotexten. Skriv en funktion, `oracle`, som tar en klartext som argument och med, utanför funktionen *globalt definierade*, krypteringsnycklar krypterar klartexten och returnerar kryptotexten. (1p)

(d) Skriv slutligen funktionen `reveal` som tar en kryptotext som argument, knäcker kryptot med hjälp av funktionen `oracle` och returnerar klartexten. (5p)

26. *Heaven or Hell*

I denna uppgift ska du datorn avgöra livsödet för en presumtiv användare.

(a) Skapa en klass `Choice` som innehåller en konstruktor och anger en icke-dold metod `decision`. (1p)

(b) I klassen `Choice`, specificera metoderna `decision` och `get_verdict`. Den första, `decision`, ska slumpmässigt tilldela den dolda variabeln `verdict` värdet `'Heaven'` eller värdet `'Hell'`. Metoden `get_verdict` ska bara returnera värdet av variablen. (5p)

(c) Sedan ska du använda klassen genom att låta `soul1` och `soul2` vara instanser av den och skriva ett program som bestämmer hur många gånger de måste exekveras för att generera samma värde, dvs båda bli `'Heaven'` eller båda bli `'Hell'`. (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`

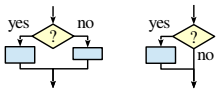
```
import monmod
```
→*access via* `monmod.nom1` …

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
```

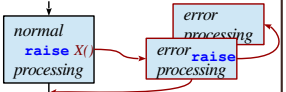
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

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*

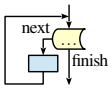
`continue` *next iteration*

`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`)*

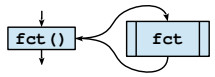
Function Call

```
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
```