Referensblad Tentamen Programmeringsteknik I 2020-03-20

Listor

- `a = [1]` skapar en lista med ett element
- `len(lst)` ger listans längd.
- `sorted_list = sorted(lst)` sorterar elementen i listan och returnerar en ny lista. Den ursprungliga listan förändras inte.
- `sorted_list = sorted(lst, key=f)` sorterar elementen i listan efter den nyckel som returneras av funktionen som ges till parametern `key`.
- `lst.pop(index)` tar bort och returnerar elementet på plats `index`. Om parametern utelämnas är `-1` underförstått. Exempelvis ger `['a', 'b', 'c'].pop(0)` resultatet `'a'`.
- `lst.append(value)` lägger till ett nytt element sist i listan.
- `del lst[index]` tar bort elementet på angivet index.

Strängar

- `len(s)` ger strängens längd.
- `s.join(lst)` sammanfogar (konkatenerar) elementen i en lista till en sträng, med strängen `s` som avskiljare mellan varje element. `s` kan vara tomma strängen `''`. Ett exempel: `'-'.join(['1', '2', '3'])` ger `'1-2-3'`.
- `s.count(subs)` räknar antalet förekomster av strängen `subs` i `s`. Exempelvis ger `'abbabba'.count('bb')` resultatet `2`, medan `'abbabba'.count('b')` ger `4`.
- `s.split()` delar upp strängen `s` i en lista med separata strängar ("ord") som separerats med blanktecken (mellanslag, tabb, radbrytningar och dylika tecken). Exempelvis ger `'hej på dig'.split()` resultatet `['hej', 'på', 'dig']`.
- `f'ABC: {a} {b:5.3f} {c:9d} {d:.2f}'` är en formatsträng som genererar en sträng från uttrycken `a`, `b`, `c` och `d`. `b` är ett flyttal som skrivs med total bredd 5 tecken och 3 decimaler, `c` ett heltal som skrivs med total bredd 9 tecken och `d` ett flyttal som skrivs utan någon specifik bredd, men med exakt två decimaler.

Sekvenser

- `range(start, stop, step)` skapar en sekvens med heltal från `start` fram till, men inte inklusive, `stop` (ett högerexklusivt intervall), med steglängd `step`. `step` kan utelämnas. Enbart `range(stop)` implicerar `start=0`. `list(range(7, 2, -1))` skapar listan `[7, 6, 5, 4, 3]`.
- `enumerate(lst, start = 0)` ger en sekvens med tupler `(index, värde)` där varje värde kommer från listan `lst` och indexen räknar uppåt från `start`. Om `start` utelämnas används 0.
- `zip(lst1, lst2)` kombinerar uppräkningselement från `lst1` och `lst2` till tupler `(värde1, värde2)`. Uppräkningen tar slut så fort någon av de ingående uppräkningarna gör det.

Var god vänd!

Lexikon

- `a = {'hej': 2}` skapar ett lexikon med en nyckel och ett värde.
- `len(lexikon)` ger antalet lagrade par av nycklar/värden i ett givet lexikon.
- `for key, value in lexikon.items():` kan användas för att loopa över alla nycklar och deras värden i ett lexikon.
- `for key in lexikon:` och `for key in lexikon.keys():` ger båda en loop över lexikonets nycklar.
- `for value in lexikon.values():` ger en loop över dess värden.

Tupler

- `a = (1, 2)` skapar en tupel med två element
- `b, c = a` packar upp en tupel (eller annan sekvens) med exakt två element till variablerna `b` och `c`.

Indexering
Gäller för olika sekvenser, som strängar, listor, tupler.

- `a[0]` ger första elementet.
- `a[-1]` ger sista elementet.
- `a[1:4]` ger element 1 till 4 (högerexklusivt), det vill säga element 1 t.o.m. 3.
- `a[1:7:2]` ger element 1 till 7 (högerexklusivt) med steglängd 2, det vill säga element 1, 3, 5.

Slumptalsgenerering

- `random.random()` ger ett slumptal (flyttal) i intervallet $[0.0, 1.0)$ (högerexklusivt).
- `random.randint(a, b)` ger ett slumpat heltal i intervallet $[a, b]$, alltså ett heltal x sådant att $a \leq x \leq b$ (högerinklusivt).

In-/utmatning

- `print(x, y, z, ..., sep=', ', end=' ')` skriver ut en serie uttryck ett efter ett, separerade av `sep` och avslutat med `end`. `sep` är som standard mellanslag och `end` är som standard `'\n'` (nyrad).
- `input(msg)` visar meddelandet `msg` och läser in en sträng från användaren. Denna sträng returneras.

Exempelklass

```
class Demo:
    """Exam demo class."""

    def __init__(self, val):
        """Store a single value as an attribute in the object."""
        self.anothername = val

    def __str__(self):
        """Represent the object as a string."""
        return 'DEMO CLASS INSTANCE: ' + str(self.anothername)

    def printme(self):
        """Print the object."""
        print(self.__str__())
```