MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Kursledare: Anders Mörtberg
Examinator: Lars Arvestad

Tentamen i
DA2005 Programmeringsteknik
7.5 hp
2026-04-17

Note: An English version of this exam starts on page 6!

- Tentan har flervalsfrågor där minst ett svarsalternativ är korrekt. Om man svarar fel eller inte har exakt antal rätta alternativ får man noll poäng på frågan.
- Del A består av flervalsfrågor (totalt 8 poäng).
- Del B består av kodfrågor med varierande poäng (totalt 12 poäng).
- Del C består av kodfrågor inspirerade av labbarna med varierande poäng (totalt 10 poäng).
- Inga import (Pythons standardbibliotek eller externa bibliotek) får användas om de inte nämns eller finns med i uppgiften. Man får använda inbyggda funktioner som len, range och map.
- All kod avser **Python 3**, dvs *inte* t.ex. Python 2.7
- **Hjälpmedel:** Ett A4 med så mycket information du vill. Du får skriva på båda sidorna.
- **Betygsgränser:** E: 15, D: 18, C: 21, B: 24, A: 27, av maximala 30.

Del A: flervalsfrågor (8p)

Var snäll samla svaren på del A på ett svarspapper.

1. Hur många tilldelningar av `True` och `False` till `x` och `y` gör `(not x and not y) or (x and y)` till `True`?

A. 0
B. 1
C. 2
D. 3
E. 4

2. Betrakta kodsnutten till höger, vad är värdet på `out` när den har körts?

A. `Hej`
B. `HHHeeejjj`
C. `HHHeej`
D. `jjjeeH`
E. `jjjeeeHHH`

```
s = ["H","e","j"]
out = ""

while s != []:
    l = len(s)
    out += l * s.pop()
```

3. Betrakta kodsnutten till höger, vad är värdet på `v` när den har körts?

A. -2
B. -1
C. 0
D. 1
E. 2

```
v = 0

for x in range(4):
    if x % 2 == 0:
        v += x
    else:
        v -= x
```

4. Vilka av följande påståenden är korrekta baserat på information från kodraden till höger?

A. A, B och C är klasser.

B. A är en klass, B och C är metoder.

C. B och C är klasser och A är en metod.

D. B och C ärver från A.

E. A ärver från B och C.

```
class A(B, C):
```

5. I vilken ordning skrivs tuplerna ut av koden till höger?

A. `(1,a) (1,b) (2,a) (2,b)`

B. `(1,b) (1,a) (2,b) (2,a)`

C. `(1,a) (2,a) (1,b) (2,b)`

D. `(2,a) (2,b) (1,a) (1,b)`

E. `(2,b) (2,a) (1,b) (1,a)`

```
for x in [1,2]:
    for y in ['a','b']:
        print((x,y))
```

6. Vad blir resultatet av `[ 3 ** x for x in range(3)]`?

A. `[0, 1, 2]`

B. `[1, 3, 6]`

C. `[1, 3, 6, 9]`

D. `[1, 3, 9]`

E. `[1, 3, 9, 27]`

7. Vilka av följande påståenden är sanna om Python?

A. Listor är muterbara

B. Listor är omuterbara

C. Strängar är muterbara

D. Funktioner kan ta in funktioner som argument

E. Funktioner kan anropa sig själva

8. Betrakta kodsnutten till höger. Vad blir resultatet av anropet `f("Hej")`?

A. `Hej`

B. `Hje`

C. `jeH`

D. `ejH`

E. `eHj`

```
def f(s):
    if s == "":
        return s
    else:
        return s[0] + f(s[1:][::-1])
```

Del B: kodfrågor (12p)

Var snäll använd ett papper till varje fråga i del B. Delfrågor får gärna lösas på samma papper.

9. Skriv en funktion `product_nonzero(ns)` som tar in en lista `ns` med heltal (**`int`**) och returnerar produkten av alla tal som inte är noll. (1p)

Exempelanvändning:

```
[In: ] print(product_nonzero([2,3,0,2]))
[Out:] 12
[In: ] print(product_nonzero([]))
[Out:] 1
```

10. Skriv en funktion `ask_numbers(n)` som frågar användaren efter `n` flyttal (**`float`**), dessa skrivs sedan ut till användaren. Programmet ska ha lämplig felhantering med hjälp av **`try`**/**`except`** så att det inte kraschar om användaren skriver in något som inte är ett tal. (3p)

Obs: för full poäng måste programmet fungera korrekt vid felaktig indata, dvs användaren ska alltid få skriva in `n` tal oavsett hur många gånger hen skriver något som inte är ett tal.

Exempelkörning av `ask_numbers(3)`:

```
Write a number: 2.42
Write a number: -3.1
Write a number: hello
That is not a number!
Write a number: 7
Thank you, you wrote: 2.42 -3.1 7.0
```

11.

A. Skriv en funktion `mask(s1,s2)` som tar in två strängar `s1` och `s2` som byter ut bokstäverna till `*` i `s1` i den ordning de förekommer i `s2`. (2p)

Exempel:

```
[In ]: print(mask('Hello, how are you Anders?', 'ehoaAr'))
[Out]: H*llo, **w *re you *nde*s?
[In ]: print(mask('Hello, how are you Anders?', 'H,a?Apabepa'))
[Out]: *ello* how *re you Anders*
```

Obs: gemener (små bokstäver) och versaler (stora bokstäver) ska spela roll för utdata. Notera även att i det första exemplet byts inte `o` ut i `you` då det redan använts för att byta ut `o` i `how`.

B. Skriv en funktion `unmask(s1,s2)` som tar in en maskad sträng `s1` och en sträng `s2` och byter ut alla `*` i `s1` mot bokstäverna i `s2`. (1p)

För enkelhetens skull ska man anta att `s2` har minst lika många bokstäver som det finns `*` i `s1`.

Exempel:

```
[In ]: print(unmask('H*llo, **w *re you *nde*s?', 'ehoaAr'))
[Out]: Hello, how are you Anders?
[In ]: print(unmask('*ello* how *re you Anders*','H,a?Apabepa'))
[Out]: Hello, how are you Anders?
```

12. På ordinarie tenta skulle man skriva funktioner för att hantera inköpslistor, låt oss utöka funktionaliteten lite.

En inköpslista representeras smidigt med en uppslagstabell från strängar till tal, t.ex:

```
{ "Avocado": 3 , "Butter": 1 , "Milk": 2 }
```

När man sedan ska betala måste det totala priset beräknas. Skriv `total_price(d,prices)` som tar in en uppslagstabell `d` som ovan, en uppslagstabell med priser för olika varor, och sedan beräknar det totala priset. Om någon vara i `d` inte finns i `prices` ska ett `ValueError` lyftas med hjälp av **`raise`**. (2p)

Exempelanvändning:

```
[In: ] cart = { "Avocado" : 3 , "Butter" : 1 , "Milk" : 2 }
[In: ] prices = { "Avocado" : 14, "Bread" : 37, "Butter" : 55, "Milk" : 20 }
[In: ] print(total_price(cart,prices))
[Out:] 137
[In: ] new_cart = { "Cheese" : 1 }
[In: ] print(total_price(new_cart,prices))
[Out:] Traceback (most recent call last):
       ...
       ValueError: The item 'Cheese' cannot be found
```

13. Inköpslistor igen. På ordinarie tenta skulle man även modellera varor som ska köpas med hjälp av klasser. Nedan följer en del av lösningen där en vara, `Item`, har ett namn och bäst före datum.

```
class Item():

    def __init__(self,name,y,m,d):
        self.name = name
        self.best_before = (y,m,d)
```

Utöka kodexemplet ovan på följande sätt:

A. Lägg till ett instansattribut `price` i `Item` för priset på varan. Man ska kunna sätta detta med konstruktorn, se exempel nedan. (1p)

Exempelanvändning:

```
[In: ] m = Item("Milk",2026,4,17,21) # Mjölk som kostar 21 kr och går ut 17/4 2026
[In: ] print(m.price)
[Out:] 21
```

B. Lägg till klassen `Fruit` som ärver `Item`, men som i sin tur lägger till instansattributen `weight` för vikt i kg och `price_per_kg` för pris per kg. Attributen som ärvs från superklassen ska sättas med hjälp av **`super()`** och priset på frukten ska då sättas till vikten gånger priset per kg. Konstruktorn för `Fruit` ska alltså bara ta in namn, bäst före datum, vikt och pris per kg. (2p)

Exempelanvändning:

```
[In: ] b = Fruit("Bananas",2026,4,17,2,24) # 2kg bananer som kostar 24 kr/kg
[In: ] print(b.price)
[Out:] 48
```

Del C: kodfrågor inspirerade av labbarna (10p)

Var snäll använd ett papper till varje fråga i del C. Delfrågor får gärna lösas på samma papper.

14. Givet en fil `infile` med ett tal per rad:

```
5
12
7
20
3
```

Skriv en funktion `averages(infile,outfile)` som öppnar filen och beräknar medelvärde som uppdateras för varje tal på varje rad. Resultatet ska skrivas till `outfile` på följande format:

```
Average 1: 5.0
Average 2: 8.5
Average 3: 8.0
Average 4: 11.0
Average 5: 9.4
```

Medelvärdena ser ut på detta sätt då första raden är bara 5 och medelvärdet är då $5/1 = 5$. På andra raden har vi läst in ett nytt tal, 12, så nya medelvärdet blir $(5 + 12)/2 = 17/2 = 8.5$. Sen läser vi in 7 och nya medelvärdet blir $(5 + 12 + 7)/3 = 24/3 = 8$, osv.

För full poäng ska man ha korrekt felhantering med hjälp av **`try`**/**`except`** så att funktionen inte kraschar om `infile` inte finns eller inte har rätt format (ett tal per rad). (4p)

15. På labb 2 skulle man skriva funktioner för att hantera hexadecimaltal. Dessa är tal i bas 16 och eftersom vi bara har tio siffror 0-9 att tillgå så kompletteras systemet med bokstäverna A-F, som får representera värdena 10-15.

Ett annat vanligt talsystem när man håller på med datorer är det binära talsystemet. Man jobbar då i bas 2 och har bara ettor och nollor. Då $16 = 2^4$ är det lätt att översätta hexadecimaltal till binära tal. Idén är följande: givet ett hexadecimaltal så kan man byta ut varje hexadecimalsiffra mot exakt fyra ettor/nollor enligt tabellen nedan (decimaltalen är bara med i tabellen för jämföra med vanliga tal, de är inte relevanta för lösningen av uppgiften):

Decimal	0	1	2	...	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	...	8	9	A	B	C	D	E	F
Binär	0000	0001	0010	...	1000	1001	1010	1011	1100	1101	1110	1111

Skriv en funktion `hex_to_bin(h)` som tar in ett hexadecimaltal och returnerar ett binärt tal. Precis som på labben ska ni använda strängar för att representera talen. Man kan anta att indata är ett korrekt hexadecimaltal (så en sträng med `0` till `F`). För full poäng får inte utdata ha några nollor längst till vänster. Detta är av samma anledning som man inte brukar ha med nollor längst till vänster när man skriver vanliga tal, dvs man skriver 42, inte 0000042. (2p)

För att göra detta lättare får ni använda följande uppslagstabell för att konvertera en hexadecimalsiffra till ett binärt tal med fyra ettor/nollor (ni behöver inte kopiera uppslagstabellen till er lösning utan ni kan använda den fritt i er kod).

```
hex_digit_to_bin = { "0" : "0000" , "1" : "0001" , "2" : "0010" , "3" : "0011"
                   , "4" : "0100" , "5" : "0101" , "6" : "0110" , "7" : "0111"
                   , "8" : "1000" , "9" : "1001" , "A" : "1010" , "B" : "1011"
                   , "C" : "1100" , "D" : "1101" , "E" : "1110" , "F" : "1111" }
```

Exempelanvändning:

```
[In: ] print(hex_to_bin("10"))
[Out:] 10000
[In: ] print(hex_to_bin("F10"))
[Out:] 111100010000
[In: ] print(hex_to_bin("0123456789ABCDEF"))
[Out:] 1001000110100010101100111100010011010101111001101111101111
[In: ] print(hex_to_bin("FEDCBA9876543210"))
[Out:] 1111111011011100101110101001100001110110010101000011001000010000
```

Obs: Eftersom nollor i början tas bort får vi inte att `hex_to_bin("10")` blir `00010000`.

16. Skriv en funktion `bin_to_hex(b)` som går åt andra hållet jämfört med förra uppgiften. För full poäng får funktionen inte krascha om antalet ettor och nollor inte är jämnt delbart med 4. (4p)

Här får man använda uppslagstabellen nedan som konverterar fyra ettor och nollor till en hexadecimalsiffra.

```
bin_digit_to_hex = { "0000" : "0" , "0001" : "1" , "0010" : "2" , "0011" : "3"
                   , "0100" : "4" , "0101" : "5" , "0110" : "6" , "0111" : "7"
                   , "1000" : "8" , "1001" : "9" , "1010" : "A" , "1011" : "B"
                   , "1100" : "C" , "1101" : "D" , "1110" : "E" , "1111" : "F" }
```

Obs: Tänk på vad som händer om längden på `b` inte är jämnt delbar med `4`, vilket t.ex. är fallet för `10000` binärt vilket motsvarar `10` i hex. För att göra konvertering bör man alltså slå upp `0001` och `0000` i uppslagstabellen, inte `1` och `0000`.

Exempelanvändning:

```
[In: ] print(bin_to_hex("10000"))
[Out:] 10
[In: ] print(bin_to_hex("111100010000"))
[Out:] F10
[In: ] print(bin_to_hex("1001000110100010101100111100010011010101111001101111101111"))
[Out:] 123456789ABCDEF
[In: ] all_digits = "1111111011011100101110101001100001110110010101000011001000010000"
[In: ] print(bin_to_hex(all_digits))
[Out:] FEDCBA9876543210
```

English translation

- The exam has multiple-choice questions where at least one answer option is correct. If you answer incorrectly or do not select the exact number of correct alternatives, you receive zero points for the question.
- Part A consists of multiple-choice questions (a total of 8 points).
- Part B consists of coding questions with varying point values (a total of 12 points).
- Part C consists of coding questions inspired by the labs, with varying point values (a total of 10 points).
- No import (Python's standard library or external libraries) may be used unless they are mentioned or included in the assignment. You may use built-in functions such as len, range, and map.
- All code refers to **Python 3**, i.e., *not* e.g. Python 2.7.
- **Allowed aids:** One A4 sheet with as much information as you want. You may write on both sides.
- **Grade thresholds:** E: 15, D: 18, C: 21, B: 24, A: 27, out of a maximum of 30.

Part A: multiple-choice questions (8 points)

Please gather your answers to part A on a single answer sheet. Do not mark your answers on this paper!

1. How many assignments of `True` and `False` to `x` and `y` make `(not x and not y) or (x and y)` evaluate to `True`?

A. 0
B. 1
C. 2
D. 3
E. 4

2. Consider the code snippet on the right. What is the value of `out` after the code has been executed?

A. `Hej`
B. `HHHeeejjj`
C. `HHHeej`
D. `jjjeeHH`
E. `jjjeeeHHH`

```
s = ["H","e","j"]
out = ""

while s != []:
    l = len(s)
    out += l * s.pop()
```

3. Consider the code snippet on the right. What is the value of `v` after the code has been executed?

A. -2
B. -1
C. 0
D. 1
E. 2

```
v = 0

for x in range(4):
    if x % 2 == 0:
        v += x
    else:
        v -= x
```

4. Which of the following statements are correct based on information from the line of code on the right?

A. A, B, and C are classes.
B. A is a class, B and C are methods.
C. B and C are classes and A is a method.
D. B and C inherit from A.
E. A inherits from B and C.

```
class A(B, C):
```

5. In what order are the tuples printed by the code on the right?

A. `(1,a) (1,b) (2,a) (2,b)`

B. `(1,b) (1,a) (2,b) (2,a)`

C. `(1,a) (2,a) (1,b) (2,b)`

D. `(2,a) (2,b) (1,a) (1,b)`

E. `(2,b) (2,a) (1,b) (1,a)`

```
for x in [1,2]:
    for y in ['a','b']:
        print((x,y))
```

6. What is the result of `[ 3 ** x for x in range(3)]`?

A. `[0, 1, 2]`

B. `[1, 3, 6]`

C. `[1, 3, 6, 9]`

D. `[1, 3, 9]`

E. `[1, 3, 9, 27]`

7. Which of the following statements are true about Python?

A. Lists are mutable.

B. Lists are immutable.

C. Strings are mutable.

D. Functions can take functions as arguments.

E. Functions can call themselves.

8. Consider the code snippet on the right. What is the result of the call `f("Hej")`?

A. `Hej`

B. `Hje`

C. `jeH`

D. `ejH`

E. `ejH`

```
def f(s):
    if s == "":
        return s
    else:
        return s[0] + f(s[1:][::-1])
```

Part B: code problems (12 points)

Please use a paper for each problem in part B. You are welcome to use the same sheet for multiple subproblems. Only write on one side!

9. Write a function `product_nonzero(ns)` that takes a list `ns` of integers (**`int`**) and returns the product of all numbers that are not zero. (1p)

Example usage:

```
[In: ] print(product_nonzero([2,3,0,2]))
[Out:] 12
[In: ] print(product_nonzero([]))
[Out:] 1
```

10. Write a function `ask_numbers(n)` that asks the user for `n` floating-point numbers (**`float`**), which are then printed to the user. The program should have appropriate error handling using **`try`**/**`except`** so that it does not crash if the user enters something that is not a number. (3p)

Note: to receive full credit, the program must function correctly for invalid input, i.e., the user should always be able to enter `n` numbers regardless of how many times they enter something that is not a number.

Example execution of `ask_numbers(3)`:

```
Write a number: 2.42
Write a number: -3.1
Write a number: hello
That is not a number!
Write a number: 7
Thank you, you wrote: 2.42 -3.1 7.0
```

11.

A. Write a function `mask(s1,s2)` that takes two strings `s1` and `s2` and replaces the letters in `s1` with `*` in the order in which they appear in `s2`. (2p)

Example:

```
[In ]: print(mask('Hello, how are you Anders?', 'ehoaAr'))
[Out]: H*llo, **w *re you *nde*s?
[In ]: print(mask('Hello, how are you Anders?', 'H,a?Apabepa'))
[Out]: *ello* how *re you Anders*
```

Note: lowercase and uppercase letters must be treated as distinct. Also note that in the first example, the `o` in `you` is not replaced since it has already been used to replace the `o` in `how`.

B. Write a function `unmask(s1,s2)` that takes a masked string `s1` and a string `s2` and replaces all `*` in `s1` with the letters in `s2`. (1p)

For simplicity, assume that `s2` contains at least as many letters as there are `*` characters in `s1`.

Example:

```
[In ]: print(unmask('H*llo, **w *re you *nde*s?', 'ehoaar'))
[Out]: Hello, how are you Anders?
[In ]: print(unmask('*ello* how *re you Anders*','H,a?Apabepa'))
[Out]: Hello, how are you Anders?
```

12. On the regular exam, one would write functions to handle shopping lists; let us extend the functionality slightly.

A shopping list is conveniently represented by a dictionary mapping strings to numbers, for example:

```
{ "Avocado": 3 , "Butter": 1 , "Milk": 2 }
```

When it is time to pay, the total price must be computed. Write `total_price(d,prices)` which takes a dictionary `d` as above, a dictionary with prices for different items, and then computes the total price. If an item in `d` is not present in `prices`, a `ValueError` should be raised using **`raise`**. (2p)

Example usage:

```
[In: ] cart = { "Avocado" : 3 , "Butter" : 1 , "Milk" : 2 }
[In: ] prices = { "Avocado" : 14, "Bread" : 37, "Butter" : 55, "Milk" : 20 }
[In: ] print(total_price(cart,prices))
[Out:] 137
[In: ] new_cart = { "Cheese" : 1 }
[In: ] print(total_price(new_cart,prices))
[Out:] Traceback (most recent call last):
       ...
       ValueError: The item 'Cheese' cannot be found
```

13. Shopping lists again. On the regular exam, one would also model items to be purchased using classes. Below is part of the solution in which an item, `Item`, has a name and a best-before date.

```
class Item():

    def __init__(self,name,y,m,d):
        self.name = name
        self.best_before = (y,m,d)
```

Extend the code example above as follows:

A. Add an instance attribute `price` to `Item` for the price of the item. This should be set using the constructor; see the example below. (1p)

Example usage:

```
[In: ] m = Item("Milk",2026,4,17,21) # Milk that costs 21 SEK and expires on 17/4 2026
[In: ] print(m.price)
[Out:] 21
```

B. Add the class `Fruit`, which inherits from `Item`, but additionally introduces the instance attributes `weight` for weight in kg and `price_per_kg` for price per kg. The attributes inherited from the superclass should be set using **`super`**`()`, and the price of the fruit should be set to the weight multiplied by the price per kg. The constructor for `Fruit` should therefore only take name, best-before date, weight, and price per kg as arguments. (2p)

Example usage:

```
[In: ] b = Fruit("Bananas",2026,4,17,2,24) # 2 kg of bananas costing 24 SEK/kg
[In: ] print(b.price)
[Out:] 48
```

Part C: coding problems inspired by the labs (10 points)

14. Given a file `infile` with one number per line:

```
5
12
7
20
3
```

Write a function `averages(infile,outfile)` that opens the file and computes a running average that is updated for each number on each line. The result should be written to `outfile` in the following format:

```
Average 1: 5.0
Average 2: 8.5
Average 3: 8.0
Average 4: 11.0
Average 5: 9.4
```

The averages look this way because the first line contains only 5 and the average is therefore $5/1 = 5$. On the second line, a new number, 12, is read, so the new average becomes $(5 + 12)/2 = 17/2 = 8.5$. Then we read 7 and the new average becomes $(5 + 12 + 7)/3 = 24/3 = 8$, and so on.

For full credit, correct error handling using **`try`**/**`except`** must be implemented so that the function does not crash if `infile` does not exist or does not have the correct format (one number per line). (4p)

15. In lab 2, you were asked to write functions to handle hexadecimal numbers. These are numbers in base 16, and since we only have the ten digits 0–9 available, the system is complemented with the letters A–F, which represent the values 10–15.

Another common number system when working with computers is the binary number system. This works in base 2 and uses only ones and zeros. Since $16 = 2^4$, it is easy to convert hexadecimal numbers to binary. The idea is as follows: given a hexadecimal number, each hexadecimal digit can be replaced by exactly four ones/zeros according to the table below (the decimal numbers are included only for comparison with ordinary numbers; they are not relevant for solving the task):

Decimal	0	1	2	...	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	...	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	...	1000	1001	1010	1011	1100	1101	1110	1111

Write a function `hex_to_bin(h)` that takes a hexadecimal number and returns a binary number. As in the lab, you should use strings to represent the numbers. You may assume that the input is a valid hexadecimal number (i.e., a string containing `0` to `F`). For full credit, the output must not contain any leading zeros. This is for the same reason that we usually do not write leading zeros in ordinary numbers, i.e., we write 42, not 0000042. (2p)

To make this easier, you may use the following dictionary to convert a hexadecimal digit to a binary number consisting of four ones/zeros (you do not need to copy the dictionary into your solution; you may use it freely in your code).

```
hex_digit_to_bin = { "0" : "0000" , "1" : "0001" , "2" : "0010" , "3" : "0011"
                   , "4" : "0100" , "5" : "0101" , "6" : "0110" , "7" : "0111"
                   , "8" : "1000" , "9" : "1001" , "A" : "1010" , "B" : "1011"
                   , "C" : "1100" , "D" : "1101" , "E" : "1110" , "F" : "1111" }
```

Example usage:

```
[In: ] print(hex_to_bin("10"))
[Out:] 10000
[In: ] print(hex_to_bin("F10"))
[Out:] 111100010000
[In: ] print(hex_to_bin("0123456789ABCDEF"))
[Out:] 100100011010001010110011111000100110101011110011011111011111
[In: ] print(hex_to_bin("FEDCBA9876543210"))
[Out:] 1111111011011100101110101001100001110110010101000011001000010000
```

Note: Since leading zeros are removed, `hex_to_bin("10")` does not become `00010000`.

16. Write a function `bin_to_hex(b)` that performs the conversion in the opposite direction compared to the previous task. For full credit, the function must not crash if the number of ones and zeros is not evenly divisible by 4. (4p)

Here you may use the dictionary below, which converts four ones and zeros into a hexadecimal digit.

```
bin_digit_to_hex = { "0000" : "0" , "0001" : "1" , "0010" : "2" , "0011" : "3"
                   , "0100" : "4" , "0101" : "5" , "0110" : "6" , "0111" : "7"
                   , "1000" : "8" , "1001" : "9" , "1010" : "A" , "1011" : "B"
                   , "1100" : "C" , "1101" : "D" , "1110" : "E" , "1111" : "F" }
```

Note: Consider what happens if the length of `b` is not evenly divisible by `4`, which is for example the case for `10000` in binary, corresponding to `10` in hexadecimal. To perform the conversion, you should therefore look up `0001` and `0000` in the dictionary, not `1` and `0000`.

Example usage:

```
[In: ] print(bin_to_hex("10000"))
[Out:] 10
[In: ] print(bin_to_hex("111100010000"))
[Out:] F10
[In: ] print(bin_to_hex("100100011010001010110011111000100110101011110011011111011111"))
[Out:] 123456789ABCDEF
[In: ] all_digits = "1111111011011100101110101001100001110110010101000011001000010000"
[In: ] print(bin_to_hex(all_digits))
[Out:] FEDCBA9876543210
```