Cvičná zkouška

Cykly a podmínky

1) Napište funkci sumTuples :: int → int, která vytvoří nové číslo zřetězením součtů dvojic čísel v původním čísle. Vstupní číslo bude mít vždy sudý počet cifer, přenosy do vyšších řádů ignorujte.

sumTuples(11223344) → 2468

sumTuples(23561191) → 5120

2) Napište funkci chooseZip :: (string → string → string) → string, která vytvoří nový řetězec zazipováním dvou vstupních řetězců. Třetí řetězec obsahuje pouze znaky ‘1’ a ‘2’, nově vzniklý řetězec bude na i-té pozici obsahovat i-tý znak z prvního řetězce, pokud je na i-té pozici ze třetího řetězce ‘1’, a analogicky pro znak ‘2’ a druhý řetězec.

chooseZip(“abcdef”, “ghijkl”, “111111”) → “abcdef”

chooseZip(“abcdef”, “ghijkl”, “121212”) → “ahcjel”

3) Napište funkci counts :: int → int, která vytvoří nové číslo zřetězením počtů opakování vedle sebe stojících stejných cifer v čísle původním, podotkněme, že na hodnotě opakující se cifry vůbec nezáleží.

counts(12225599999) → 1325

counts(56722233) → 11132

4) Napište funkci waveSum :: int → int, která ke každému řádu čísla přičte řád o jedna menší. Přenosy ignorujte, při výpočtu číslo procházejte zprava.

waveSum(1111) → 4321

waveSum(138126) → 107986

Datové struktury

1) Napiště funkci counts :: int → string, která vytvoří nové číslo zřetězením počtu výskytů cifer ve vstupním čísle.

counts(12345) → ‘0111110000’

counts(1612263) → ‘0221002000’

2) Napište funkci justOnce :: [(int, string)] → [(int, string)], která na vstupu dostane seznam, ve kterém se některé dvojice mohou opakovat, a vrátí seznam dvojic bez opakování. Na pořadí dvojic ve výstupu nezáleží.

justOnce([(1,’a’),(1,’a’),(1,’a’),(1,’a’)]) → [(1,’a’)]

justOnce([(1,’a’),(2,’b’),(1,’a’),(3,’c’)]) → [(1,’a’),(1,’a’),(3,’c’)]

3) Napište funkci tupleCount :: string → [(string, int)], která spočítá výskyty dvojic ve vstupním řetězci. Předpokládejte, že má vstupní řetězec sudou délku, na pořádí dvojic nezáleží.

tupleCount(‘aabbababaa’) → [(‘aa’, 2), (‘bb’, 1), (‘ab’, 2)]

Čtení kódu

1) Popište, co program dělá

```
def f(x):
  y = 0
  while x >= 0:
    y += 1
    x -= 4
  return y
```

2) Najděte a opravte a okomentujte chyby v programu počítajícím ciferný součet vstupního čísla

```
def f(x):
  y = 0
  while x > 0:
    y += y % 10
    x -= 10
  return x
```

3) Popište, co program dělá

```
def f(x):
  if x % 2 == 2:
    return 13
  if x % 2 == 1:
    for i in range(x / 3):
      return i
  return x // 2
```

4) Najděte a opravte a okomentujte chyby v programu vytvářejícím pyramidu jedniček

```
def f(x):
  y = [1]
  z = []
  for i in range(x):
    z.append(y)
    y.append(x)
  return z
```

Pyramida jedniček (typově [[int]]) vypadá pro vstup 4 takto:

```
[[1],
 [1,1],
 [1,1,1],
 [1,1,1,1]]
```

Teoretické otázky

Najděte a opravte chyby v těchto tvrzeních

Python je staticky typovaný interpretovaný jazyk, kód napsaný v Pythonu není přenositelný na jiné zařízení. Každou proměnnou musím vždy inicialisovat, a to i proměnné, které deklaruji v hlavičce funkcí. Seznam v Pythonu smí obsahovat jen prvky stejného datového typu, zatímco v ntici mohou být prvky různých typů. Při předávání čísel do funkce se číslo kopíruje a tak nemůže dojít k aliasingu. Operátory se v Pythonu zapisují infixově, u funkcí si můžu vybrat mezi infixem a prefixem.

Stručně odpovězte na otevřené otázky

Jak se od sebe liší tyto dva zápisy?
x[1][2]
x[1,2]

Jak můžu zkopírovat pole?

Jaký je rozdíl mezi x.sort() a sorted(x)?

K čemu slouží klíčové slovo in?

Jak můžu zabránit aliasingu?

Jak můžu v aritmetickém výrazu změnit pořadí vyhodnocování?

Jaký je rozdíl mezi línou a striktní vyhodnocovací strategií?

Co se stane, když se pokusím vyhodnotit tento kód a proč?
x = True
x = 4