OSNOVE PROGRAMIRANJA

PISMENI ISPIT / januarski rok školske 2016/17 godine.

STUDENT: _____

UPUTSTVO:

- Ispit traje 2 sata
- Na ispitu se mogu koristiti samo papir i olovka
- Upotreba mobilnih telefona, kompjutera i tableta tokom ispita nije dozvoljena
- Ispit se sastoji od 5 zadataka
- Svaki rešeni zadatak donosi 20 poena
- Student koji ostvari 50 ili više poena položio je ispit

ZADATAK 1 (20 poena)

Napišite program koji traži od korisnika da unese lozinku. Ako unese ispravnu lozinku program mu saopštava da se ulogovao u sistem. U suprotnom program traži da ponovo unese lozinu. Korisnik ima samo pet pokušaja za unos lozinke, a posle toga mu program saopštava da je izbačen iz sistema.

ZADATAK 2 (20 poena)

Tražite od korisnika da unese 10 rezultata nekog testa. Napišite program koji radi sledeće:

- Štampa najveći i najmanji rezultat.
- Štampa prosečan rezultat.

ZADATAK 3 (20 poena)

Napišite program koji uzastopno traži od korisnika da unese proizvod i njegovu cenu. Smestite sve unete vrednosti u rečnik čiji je ključ naziv proizvoda a vrednost cena proizvoda. Kada korisnik završi sa unosom podataka za rečnik, dajte mu mogućnost da uzastopno unosi ime proizvoda pa štampajte njegovu cenu ili poruku da proizvod nije u rečniku.

ZADATAK 4 (20 poena)

Napišite funkciju slucajan_broj koja vraća slučajan broj između 1 i 100 koji je deljiv i sa 2 i sa 5.

ZADATAK 5 (20 poena)

Napišite klasu Student koja ima atribute: broj_indeksa, ime, polozeni_ispiti. Atribut broj_indeksa je string. Atribut ime je string koji sadrži prezime i ime studenta. Atribut polozeni_ispiti je rečnik čiji su ključevi nazivi predmeta a vrednosti postignute ocene. Klasa treba da sadrži metode: broj_položenih ispita kojom se vraća broj položenih ispita, lista_položenih ispita kojom se vraća lista položenih ispita, kao i metodu polozio_ispit(naziv_ispita,ocena) kojom se u rečniku polozeni_ispiti dodaje novi član.

Python Language & Syntax Cheat Sheet

Python is white-space dependent; code blocks are indented 4 spaces (not tabs)

Variable Assignment

```
integer = 1
string = "string"
unicode_string = u"unicode string"
mutli_line_string = """ multi-line
        string
        """
tuple = (element1, element2, element3, ...)
list = [ element1, element2, element3, ... ]
dictionary = { key1 : value1, key2 : value2, ... }
dictionary[key] = value
class_instance = ClassName(init_args)
```

Frequently Used Built-in Types

True	**False**	**None**
str	**unicode**	**int**
float	**list**	**dict**

Other than **True**, **False** and **None**, these can also be used as functions to explicitly cast a value to that type

Functions

```
def function_name(arg1, arg2,
                  keyword1=val1, keyword2=val2, ...):
    <function body>
    return return_value
```

e.g.

```
def my_function(x, y, z=0):
    sum = x + y + z
    return sum
```

```
my_function(1, 2) → 3
my_function(1, 2, 3) → 6
my_function(1, 2, y=4) → 7
```

Classes

```
class ClassName(SuperClass):
    class_variable = static_value
    def __init__(self, value1, <...>):
        self.instance_variable1 = value1
        self.instance_function()
    def instance_function(self, arg1, <...>):
        <function body>
        return return_value
```

e.g.

```
class MyClass(object):
    offset = 1
    def __init__(self, value):
        self.value = value
    def get_offset_value(self):
        return MyClass.offset +
               self.value
```

```
MyClass.offset → 1

c = MyClass(2)
c.value → 2
c.get_offset_value() → 3
```

Imports

```
import module
from module import class, function, variable
```

Frequently Used String Manipulations

string1 + *string1*	"str" + "ing" → "string"
"%s%s" % (*string1*, *string2*)	"%s%s" % ("s", "g") → "sg"
string.split("*delim*", *limit*)	"s/g".split("/") → ["s", "g"]
string.strip()	" string ".strip() → "string"
string.startswith(*"prefix"*)	"str".startswith("s") → True
substring in *string*	"str" in "string" → True

```
print string
```

List Comprehension

```
[ value for value in list if condition ]
```

e.g.

```
[x for x in [1,2,3,4,5,6,7,8,9] if x % 2 == 0] → [2,4,6,8]
```

Accessing Variable Values

```
value = dictionary[key]
value = dictionary.get(key, default_value)
value = list[index]                  e.g. [5,6,7][2] → 7
value = string[start:end]            e.g "string"[0:3] → "str"
value = list[start:end]              e.g. [1,2,3][1:2] → [2]
value = ClassName.class_variable
value = class_instance.instance_variable
value = class_instance.function(args)
```

Comparisons

value1 == *value2*	"str" == "str" → **True**
value1 != *value2*	"str" != "str" → **False**
value1 < *value2*	1 < 2 → **True**
value1 <= *value2*	2 <= 2 → **True**
value1 > *value2*	2 > 3 → **False**
value1 >= *value2*	3 >= 3 → **True**
value **is** [**not**] **None**	
value **in** *list*	1 in [2,3,4] → **False**

isinstance(*class_instance*, *ClassName*)

Basic Arithmetic

```
i = a + b        i = a - b
i = a / b        i = a * b
i = a % b        e.g. 11 % 3 → 2
```

Comments

```
"""                  # Line Comment
    Multi-line comment
"""
```

Control Flow

```
if conditional:
    <body>
elif conditional:
    <body>
else:
    <body>
```

```
for value in list:
    <body>
    continue
    break
```

```
while conditional:
    <body>
    continue
    break
```

e.g.

```
if i == 7:
    print "seven"
elif i == 8:
    print "eight"
else:
    print str(i)
```

```
for i in [1, 2, 3, 4]:
    if i == 2: continue
    if i == 3: break
    print i
```

```
while True:
    print "infinity"
```

Exceptions

```
try:
    <body>
    raise Exception()
except Exception as e:
    <exception handling>
finally:
    <clean-up>
```

e.g.

```
try:
    database.update()
except Exception as e:
    log.error(e.msg)
    database.abort()
finally:
    database.commit()
```

File & Path Manipulation

```
import os    # import the os module first
os.path.join(path_segment1, path_segment2, ...)
os.path.exists(path)
os.listdir(directory_path)
os.remove(file_path)
os.rmdir(directory_path)
file = open(path, "rw")
file.read()
string.write("string")
```

by Cottage Labs (http://cottagelabs.com)
for Dev8D (http://www.dev8d.org/)