Canadian Computing Competition

Introduction to CCC – Junior Level

http://www.cemc.uwaterloo.ca/contests/computing.html

Test basic programming skills, logic, and learn how to write a program to solve problems
Junior Level

Registration before the deadline

The topics that CCC covers

Test Grading and Our Target

We will be using python to write the test

HOW TO THINK LIKE A COMPUTER SCIENTIST – Python Review

1. Functions

Its purpose is to help organize programs into chunks that match how we think about the solution to the problem.

Syntax definition:

2. Selections

a) Boolean values and Boolean expressions:

b) Logical operators:

c) if-else statement:

d) Nested conditions:

e) Chained conditions:

f) Boolean functions:

```
"""
if condition practice
"""

def ticket_price(age):
        """
        age: int -> float
        return the ticket price for a person with age in years.
        Seniors 65 and over pay 4.75, kids 12 and under pay 4.25 and everyone else pays 7.5
        >>> ticket_price(7)
        4.25
        >>> ticket_price(21)
        7.5
        """
        pass


def different_types(obj1, obj2):
    """Return True iff obj1 and obj2 are of different types.

    >>> different_types(3, '3')
    True
    >>> different_types(108.0, 3.14)
    False
    """
    pass
```

3. Iterations

a) For loop:

b) While loop:

```
"""
loops practice
"""

def all_harrypotter(s):
        """
        return true if every character in s appear in harrypotter
        >>> all_harrypotter('hao')
        True
        >>> all_harrypotter('terror')
        True
        >>> all_harrypotter('happy')
        False
        """
        pass


def find_letter_n_times(string, letter, n):
        """
        return the smallest substring of s starting from index 0 that contains n occurances of letter
        no case-sensitive, letter appears at least n times
        >>> find_letter_n_times('UforseEducation', 'u', 2)
        'UforseEdu'
        """
        pass


def generate_table():
        """
        use for loops to generate a table
        """
        pass
```

4. Strings

a) Working with characters of string

Code Example:

```
school = “Uforse Education Inc”
m = school[3]
>>> print(m)
```

b) String methods:

upper()
lower()
capitalize()
strip()
lstrip()
rstrip()
count()
replace()
find()
rfind()
index()

c) String length:

d) String slice:

Code Example:

```
singers = "Peter, Paul, and Mary"
>> print(singers[0:5])

>> print(singers[7:11])

>> print(singers[17:21])
```

e) Strings are immutable

f) String traversal:

How to use while loop to traverse the string?

g) String concatenation:

5. List

a) list items can be different types

```
alist = [3, 67, "cat", [56, 57, "dog"], [ ], 3.14, False]
>>> print(alist[5])
```

b) list length:

c) List concatenation:

d) List slices: alist

e) Lists are mutable

Clone a list if you want to modify but keep the original

```
a = [1,3, 4]
b = a
>>> print(a==b)

>>> print(a is b)

b = a[:]
>>> print(a == b)

>>> print(a is b)
```

f) List repetition

```
origList = [45, 76, 34, 55]
>>> print(origList * 3)

fruit = ["apple", "orange", "banana", "cherry"]
>> print([1, 2] + [3, 4])

>> print(fruit + [6, 7, 8, 9])

>> print([0] * 4)
>> print([1, 2, ["hello", "goodbye"]] * 2)
```

g) List methods:

append()
insert()
pop()
sort()
reverse()
index()
count()
remove()

h) List append vs Concatenate

```
origList = [45, 32, 88]
```

```
origList.append(“cat”)
>>> origList

origList = [45, 32, 88]
origList + [“cat”]
>>> origList
```

i) Nested list

```
nested = ["hello", 2.0, 5, [10, 20]]
innerlist = nested[3]
>>> print(innerlist)

item = innerlist[1]
>>> print(item)

>>> print(nested[3][1])
```

j) List and Strings:

```
aList = aString.split()
aString = ‘ ’.join(aList)

song = "The rain in Spain..."
wds = song.split()
>>> print(wds)

wds = song.split('ai')
>>> print(wds)

aString = “hello world”
>>> list(aString)

a = "banana"
b = "banana"
>>> print(a is b)

>>> print(a == b)


a = [81, 82, 83]
b = [81, 82, 83]
>>> print(a is b)

>>> print(a == b)
```

k) Tuples

List is mutable, but tuples are immutable

Tuple as return value

julia = ("Julia", "Roberts", 1967, "Duplicity", 2009, "Actress", "Atlanta, Georgia")

```
>>> print(julia[2])
>>> print(julia[2:6])

>>> print(len(julia))

>>> for field in julia:
>>>     print(field)

julia = julia[:3] + ("Eat Pray Love", 2010) + julia[5:]
>>> print(julia)
```

```
"""
list practice
"""

def biggest_difference(nums1, nums2):
        """
        return the greatest absolute difference between numbers at corresponding positions in nums1
and nums2
        >>> biggest_difference([1, 2, 3], [6, 8, 10])
        7
        >>> biggest_difference([1, -2, 3], [-6, 8, 10])
        10
        """
        pass


def sumUntilEven(lst):
        """
        Sum all elements in a list up to but not including the first even number
        >>> sumUntilEven([1, 3, 4, 5, 6])
        4
        >> sumUntilEven([11, 13, 5, 3, 2, 8])
        32
        """
        pass
```

6. Dictionary

Dictionary is a mapping from a key to a value

a) Dictionary methods:
   keys()
   values()
I tems()
   get()
b) Traversal:

c) Dictionaries are mutable
   clone of dictionary:
   opposites = {'up': 'down', 'right': 'wrong', 'true': 'false'}
   alias = opposites

   >>> print(alias is opposites)

   alias['right'] = 'left'
   >>> print(opposites['right'])

```
"""
dictionary practice
"""

def build_placements(shoes):
    """Return a dictionary where each key is a company and each value is a
    list of placements by people wearing shoes made by that company.

    >>> build_placements(['Saucony', 'Asics', 'Asics', 'NB', 'Saucony', \
    'Nike', 'Asics', 'Adidas', 'Saucony', 'Asics'])
    {'Saucony': [1, 5, 9], 'Asics': [2, 3, 7, 10], 'NB': [4], 'Nike': [6], 'Adidas': [8]}
    """
    pass
```