Introduction to Information Science and Technology

(SI100B Python Part)

Fu Song
School of Information Science and Technology
ShanghaiTech University

Review

Two's complement representation：

To represent a negative number x:

1. start with the fixed-size binary representation of |x|
2. invert every bit
3. add 1 to the result

Question: -20= 0b? (8-bits)

00010100 => 11101011 +1 =11101100

0b1100 1000= ?

56

Outlines

1. Constants, variables and expressions
2. Control flow
3. Compound data

Constant and Variables

Constant:

- Number: int, float, complex

  1, 2, 3.14, 2+2j

- String

  “name”, “age”

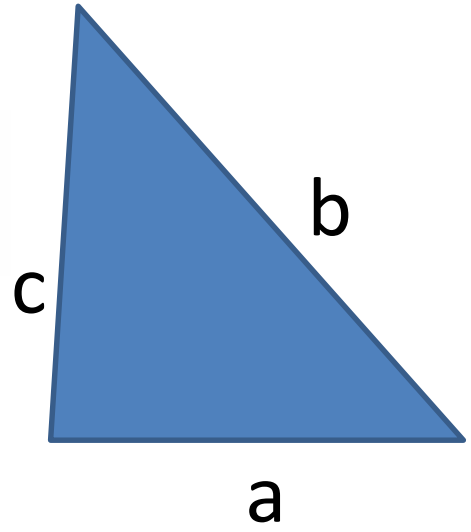
Variable: used to record values for later use

x = 1, pi =3.14

Constant and Variables

Compute area of a triangle:
Suppose a,b and c are the side lengths of the triangle

$$Area\ \Delta ABC = \sqrt{s(s-a)(s-b)(s-c)} \quad where\ \ s = \frac{a+b+c}{2}$$



s is used 4 times,

- It is better to record the value of s

Variable is a mechanism to record value

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> s = (3 + 4 + 5)/2
>>> area = math.sqrt(s * (s - 3) * (s - 4) * (s - 5))
>>> area
6.0
>>>
```

Using packages

import package

- math package provides lost of math functions , like sqrt, sin, cos, …
- dir(math) checks all the provided functions in Math
- import math imports the package to use functions defined in math package
- math.func uses function func from math, e.g. math.sqrt

Variable and Expression

1. A **variable** is a sequence of characters starting with a letter

   s, area, ... [a-zA-Z_][0-9a-zA-Z_]*

2. **Expression**: composition of variables and numbers using operators and functions

   (3+4+5)/2, s *(s-3) * (s-4) * (s-5)

   math.sqrt(s *(s-3) * (s-4) * (s-5))

3. **Assignment**: variable = expression

   s = (3+4+5)/2

   area = math.sqrt(s *(s-3) * (s-4) * (s-5))

Keywords

1. Keywords are preserved in Python
2. **Cannot** be a user-defined variable or function

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

```
>>> False = (3+4+5)/2
SyntaxError: can't assign to keyword
>>>
```

Keywords

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> from Lib import keyword
>>> keyword.iskeyword( 'if' )
True
>>> keyword.iskeyword( 'area' )
False
>>> keyword.iskeyword( 'True' )
True
>>> keyword.iskeyword( 'true' )
False
>>>
```

Case sensitive

Function

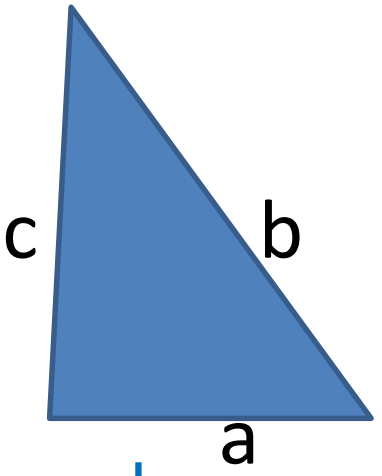
Python provides lots of useful functions defined by someone else, e.g., sqrt

Can I define my functions ?

Compute area of a triangle:

Suppose a,b and c are the side lengths of the triangle

$$Area\ \Delta ABC = \sqrt{s(s-a)(s-b)(s-c)} \quad where\ \ s = \frac{a+b+c}{2}$$



Define a function Area(a,b,c) to compute the area of a triangle

- Keyword: def
- Function name: Area
- Input parameters: a,b,c
- Result: return
- Indentation (SyntaxError)

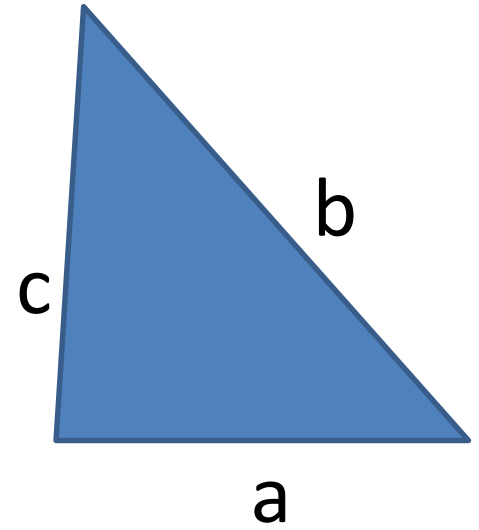
```
>>> def Area(a, b, c):
        s = (a + b + c) /2
        area = math.sqrt(s * (s-a) * (s-b) * (s-c))
        return area

>>> Area(3, 4, 5)
6.0
>>> Area(5, 5, 6)
12.0
>>>
```

If statement

```
>>> def Area(a, b, c):
        s = (a + b + c) /2
        area = math.sqrt(s * (s-a) * (s-b) * (s-c))
        return area

>>> Area(3, 4, 5)
6.0
>>> Area(5, 5, 6)
12.0
>>>
```



What will happen if we compute Area(2,2,5)?

How to solve?

```
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    Area(2,2,5)
  File "<pyshell#9>", line 3, in Area
    area = math.sqrt(s * (s-a) * (s-b) * (s-c))
ValueError: math domain error
```

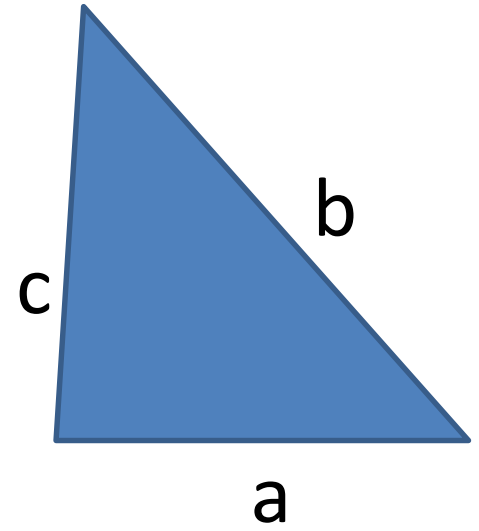
Why?

Math.sqrt(4.5 * 2.5 * 2.5 * (-0.5))

If statement

```
>>> def Area(a, b, c):
        s = (a + b + c) /2
        area = math.sqrt(s * (s-a) * (s-b) * (s-c))
        return area

>>> Area(3, 4, 5)
6.0
>>> Area(5, 5, 6)
12.0
>>>
```



What will happen if we compute Area(2,2,5)?

Solution: we need to check the condition of a, b, c

1. a + b > c
2. a + c > b
3. b + c > a

If statement

If-else: Selects statements based on a condition

Form 1

```
if condition:
        statements
```

Form 2

```
if condition1:
        statements
elif condition2:
        statements
      ……
else:
        statements
```

```
>>> import math
>>> def Area(a, b, c):
        if (a + b <=c):
                print("It is not a triangle")
                return None
        elif (a + c <=b):
                print("It is not a triangle")
                return None
        elif (b + c <=a):
                print("It is not a triangle")
                return None
        else:
                s = (a + b + c) /2
                area = math.sqrt(s * (s-a) * (s-b) * (s-c))
                return area


>>> Area(3, 4, 5)
6.0
>>> Area(2, 2, 5)
It is not a triangle
```

If statement

If-else: Selects statements based on a condition

Form 1

```
if condition:
    statements
```

Form 2

```
if condition1:
    statements
elif condition2:
    statements
  ……
else:
    statements
```

```
>>> import math
>>> def Area(a, b, c):
        if (a + b <=c) or (a + c <=b) or (b + c <=a):
            print("It is not a triangle")
            return None
        s = (a + b + c) /2
        area = math.sqrt(s * (s-a) * (s-b) * (s-c))
        return area

>>> Area(2,2,5)
It is not a triangle
>>> Area(3,4,5)
6.0
>>>
```

If statement (cond.)

If-else: Selects statements based on a condition

Can be nested

Form 1

```
if condition:
        statements
```

Form 2

```
if condition1:
        statements
elif condition2:
        statements
    ......
else:
        statements
```

```
>>> import math
>>> def Area(a, b, c):
        if (a + b >c):
                if (a + c >b):
                        if (b + c > a):
                                s = (a + b + c) /2
                                area = math.sqrt(s * (s-a) * (s-b) * (s-c))
                                return area
                        else: print("It is not a triangle")
                else: print("It is not a triangle")
        else: print("It is not a triangle")
        return None

>>> Area(3, 4, 5)
6.0
>>> Area(2, 2, 5)
It is not a triangle
```

You can write some functions to solve real problems

Let us try!

$$\sum_{1}^{n} = 1 + 2 + 3 + \cdots + n$$

$$\sum_{1}^{n} = 1 + 2 + 3 + \cdots + n = n(1 + n)/2$$

```
>>> def Sum(n):
        s = n * (1 + n)/2
        return s

>>> Sum(10)
55.0
```

Let us try!

$$\sum_{1}^{n} = 1^2 + 2^2 + 3^2 + \cdots + n^2$$

$$\sum_{1}^{n} = 1^2 + 2^2 + 3^2 + \cdots + n^2 = n(n+1)(2n+1)/6$$

```
>>> def Sum2(n):
        s = n * (n + 1) * (2 * n + 1) / 6
        return s

>>> Sum2(10)
385.0
```

How about those?

$$\sum_{1}^{n} = 1^3 + 2^3 + 3^3 + \cdots + n^3$$

$$\sum_{1}^{n} = 1^4 + 2^4 + 3^4 + \cdots + n^4$$

$$\sum_{1}^{n} = 1^5 + 2^5 + 3^5 + \cdots + n^5$$

$$\cdots$$

$$\sum_{1}^{n} = 1^k + 2^k + 3^k + \cdots + n^k$$

For loops

Perform an action/task more than once

- Pour a cup of coffee for each guest
- Wash the dishes until they are all clean
- Make a name card for each guest attending a party

$$\sum_{1}^{n} = 1^3 + 2^3 + 3^3 + \cdots + n^3$$

$$\sum_{1}^{n} = 1^4 + 2^4 + 3^4 + \cdots + n^4$$

$$\sum_{1}^{n} = 1^5 + 2^5 + 3^5 + \cdots + n^5$$

$$\cdots$$

$$\sum_{1}^{n} = 1^k + 2^k + 3^k + \cdots + n^k$$

for variable in sequence of items:
    statements

```
>>> words = ['My', 'name', 'is', 'Fu', 'Song']
>>> for w in words:
        print(w,len(w))


My 2
name 4
is 2
Fu 2
Song 4
```

```
>>> def Sum(n, k):
        s = 0
        for i in range(1,n+1):
            s = s + i ** k
        return s

>>> Sum(10, 2)
385
```

While loops

Perform an action/task more than once

- Pour a cup of coffee for each guest
- Wash the dishes until they are all clean
- Make a name card for each guest attending a party

$$\sum_{1}^{n} = 1^3 + 2^3 + 3^3 + \cdots + n^3$$

$$\sum_{1}^{n} = 1^4 + 2^4 + 3^4 + \cdots + n^4$$

$$\sum_{1}^{n} = 1^5 + 2^5 + 3^5 + \cdots + n^5$$

$$\cdots$$

$$\sum_{1}^{n} = 1^k + 2^k + 3^k + \cdots + n^k$$

while condition:
    statements

```
words = ['My', 'name', 'is', 'Fu', 'Song']
>>> i = 0
>>> while i< len(words):
        print(words[i],len(words[i]))
        i = i + 1


My 2
name 4
is 2
Fu 2
Song 4
```

```
>>> def Sum(n, k):
        s = 0
        i = 1
        while i<= n:
            s = s + i ** k
            i = i + 1
        return s

>>> Sum(10, 2)
385
```

While vs. for

- For loop knows the number of times of the loop
  - ✓ based on a generator or a sequence of items
  - ✓ always terminate
- While loop does not know the number of times
  - ✓ based on a condition (True or False)
  - ✓ may not terminate (infinite)
- Any for loop can be converts into while loop
- It is better to use for loop when it is possible

While vs. for

```
>>> n = 4
>>> for i in range(n):
        print(i)
        n = n + 1


0
1
2
3
>>>
```

```
>>> n = 4
>>> i = 0
>>> while i < n:
        print(i)
        i = i + 1
        n = n + 1


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

Break

- To terminate the loop before the specified number or the condition becomes False
- break statement

```
>>> n = 4
>>> i = 0
>>> while i < n:
        if (i==6):
                i = i + 1
                break
        print(i)
        i = i + 1
        n = n + 1


0
1
2
3
4
5
>>>
```

Continue

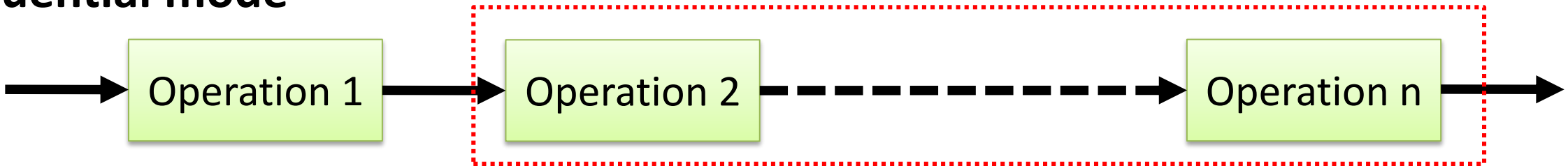
- To skip some iteration when a condition holds, but continue next iteration of the loop
- continue statement

```
>>> n = 4
>>> i = 0
>>> while i < n:
        if (i==6):
                i = i + 1
                continue
        print(i)
        i = i + 1
        n = n + 1
```

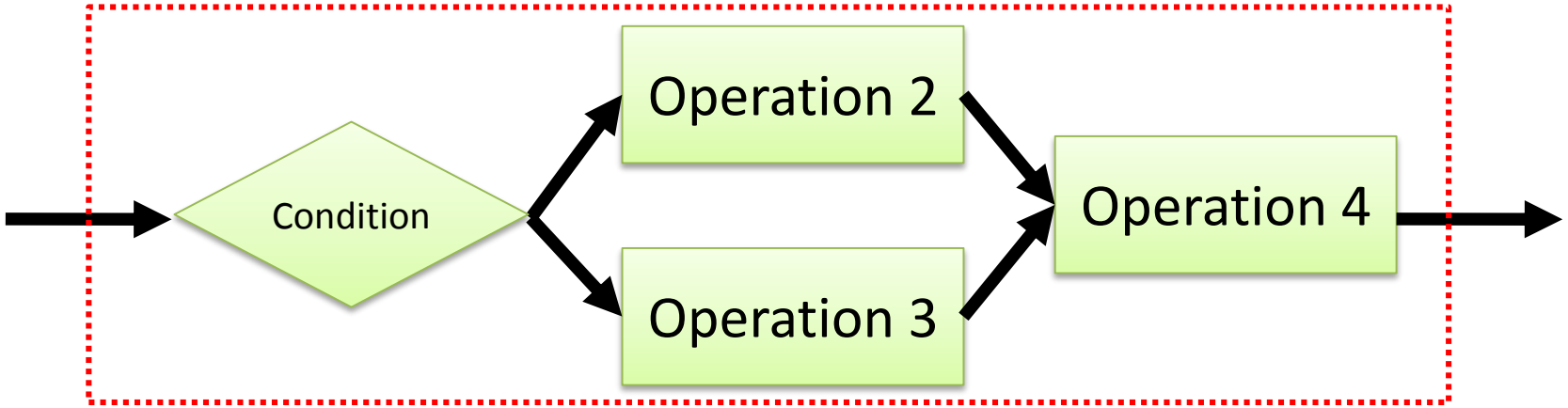
```
0
1
2
3
4
5
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

Control flow and Operations

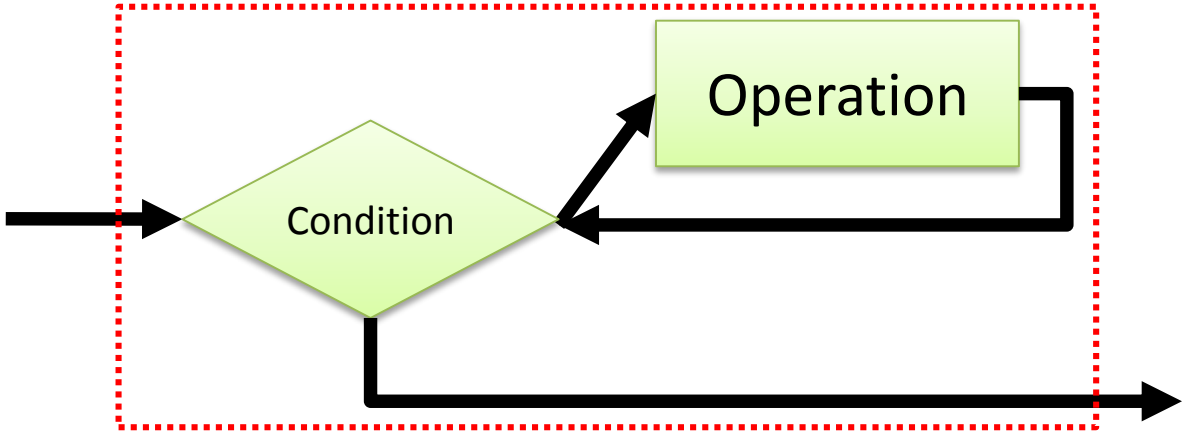
Sequential mode



Select (if-else)



Repeat(while/for loop)



Repetition

- One of the problems with code is you're frequently doing the same thing over and over again
  - The same few lines of code
  - The same tasks
  - The same operations
    - Again, and again, and again…

How to reuse code?

1. For/while loops
2. Functions
3. Classes
4. Modules
5. Packages

What is a function?

- Function:
  - (Noun) A reusable section of code with a name that does something
  - Sometimes called a method

- You have already used functions!

  print

  sqrt

  Area and Sum (user-defined)

Why create functions?

- Code reuse
  - You are doing the same thing over and over again
- Simplify your code
  - Functions have names to define what they do
  - Breakdown complex blocks of code
- Easier to make changes
  - If it's only been written once, you only have to update it once

How do you create a function?

def funcName(arg_1,…., arg_n) :
    statements

def funcName(arg_1,…., arg_n) :
    statements
    return expr

```
>>> def Area(a, b, c):
        s = (a + b + c) /2
        area = math.sqrt(s * (s-a) * (s-b) * (s-c))
        return area

>>> Area(3, 4, 5)
6.0
>>> Area(5, 5, 6)
12.0
>>>
```

Call function: funcName($expr_1$,…,$expr_n$)

1. Compute value of $expr_i$ from left to right
2. Assign results of $expr_1$,…, $expr_n$ to arg_1,…,arg_n
3. Execute statements in function definition
4. Terminate when return executes, return the value of expr or None
5. Or terminate in where is no more statements to execute

Function calls function

```
>>> def Sum(n, k):
        s = 0
        for i in range(1,n+1):
            s = s + i ** k
        return s

>>> Sum(10,2)
385
```

$$\sum_{1}^{n} = 1^3 + 2^3 + 3^3 + \cdots + n^3$$

$$\sum_{1}^{n} = 1^4 + 2^4 + 3^4 + \cdots + n^4$$

$$\sum_{1}^{n} = 1^5 + 2^5 + 3^5 + \cdots + n^5$$

$$\cdots$$

$$\sum_{1}^{n} = 1^k + 2^k + 3^k + \cdots + n^k$$

```
>>> def Sum(n,k):
        if n==1: return 1
        else: return n**k + Sum(n-1,k)

>>> Sum(10,2)
385
...
```

Function—Default argument Values

```
>>> def Sum(n,k):
        if n==1: return 1
        else: return n**k + Sum(n-1,k)


>>> Sum(10,2)
385
```

```
>>> def Sum(n,k=2):
        if n==1: return 1
        else: return n**k + Sum(n-1,k)


>>> Sum(10)
385
>>> Sum(10,3)
3025
>>> Sum(10,2)
385
```

Function—Default argument Values

The default value is evaluated only once

```
>>> def f(a, L=[]):
        L.append(a)
        return L

>>> f(1)
[1]
>>> f(2)
[1, 2]
>>> f(3)
[1, 2, 3]
>>> f(4, ['a','b'])
['a', 'b', 4]
>>> f(5)
[1, 2, 3, 5]
```

Function—Default argument Values

Note: the default values are only for the right-most arguments

```
>>> def f(a, L = [], b ):
        L.append(a)

SyntaxError: non-default argument follows default argument
```

Anonymous functions

Lambda expressions allows us to define anonymous functions

Note: syntactically restricted to a single expression

```
>>> def powAddSub(a):
        if a=="+": return lambda a,b: a**2 + b**2
        if a=="-": return lambda a,b: a**2 - b**2


>>> powAdd = powAddSub("+")
>>> powAdd(2,2)
8
>>> powSub = powAddSub("-")
>>> powSub(2,2)
0
```

You have learnt all the basic operations

Summary

1. Constants, variables and expressions
2. Control flow:
   - Assignment
   - If-else,
   - For/while loop
   - Function/lambda
3. Compound data:
   - Numeric types: int, long, float, complex, Bool
   - Sequence types: list, tuple, range, str
   - Mapping types: dict
   - Set types: set, frozen set
   - None type: type(None)

Quiz

Q1: Write the math function of the following program, e.g., foo= ….

```
def foo(n): # n is an integer
    s = 1
    for i in range(1, n+1):
        x = lambda a, b: a * b
        s = x(s, i)
    return s
```

Q2: Write the output of bar(12):

```
def bar(n): # n is an integer
    while n != 1:
        print(n)
        if n % 2 == 0: n = n / 2
        else: n = n * 3 +1
```

Solutions

Q1: Write the math function of the following program, e.g., foo(n) = ….

```
def foo(n): # n is an integer
    s = 1
    if n<0: return 0
    for i in range(1, n+1):
        x = lambda a, b: a * b
        s = x(s, i)
    return s
```

$$foo(n) = \begin{cases} n!, & \text{if } n>=0 \\ 0. & \text{else} \end{cases}$$

Q2: Write the output of bar(12):

```
def bar(n):
    while n != 1:
        print(n)
        if n % 2 == 0: n = n / 2
        else: n = n * 3 +1
```

12
6.0
3.0
10.0
5.0
16.0
8.0
4.0
2.0