Structures of Simple Python and C Program

A Python Program

```
# sample1.py : Finds GCD
s = input("Enter a +ve integer: ")
l = input("Enter another +ve integer: ")
print "gcd(",s,",",l,") = ",
while(s !=0):
    s, l = l%s, s
print l
```

Note

- Comment: `# sample.py` : Finds GCD1
- Input: `input("Enter a +ve integer: ")`
- Assignment:
  `s = input("Enter a +ve integer: ")`
- Loop: `while(s !=0):`
- Loop Body (tuple): `s, l = l%s, s`
- Output: `print "gcd(",s,",",l,") = ",`

C Program

```
#include <stdio.h>
int main() { // sample1.c
    int l, s, rem;
    printf("Enter two +ve integers\n");
    scanf("%d%d", &l, &s);
    printf("HCF(%d, %d) = ", s, l);
    while(s){
        rem = l%s; l = s; s = rem;
    }
    printf("%d\n", l);
    return 0;
}
```

Note

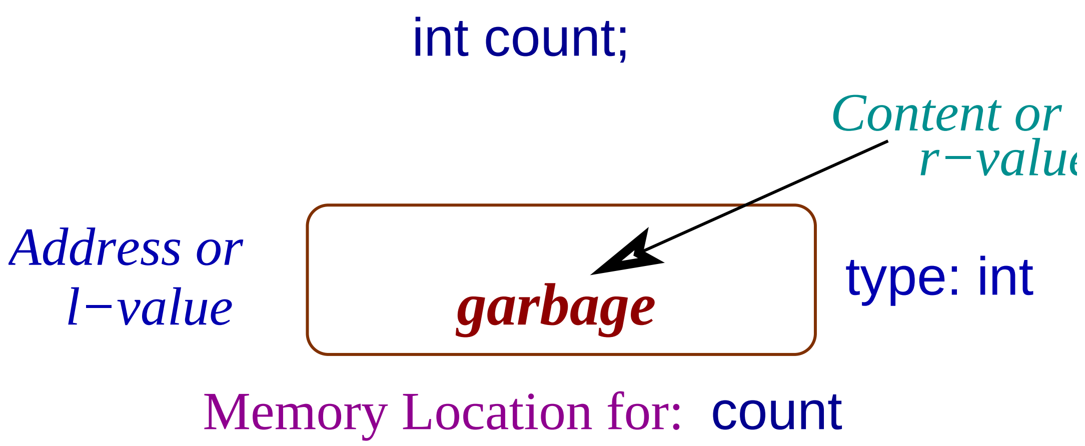
- CPP: `#include <stdio.h>`
- Function: `int main(){  }`
- Comment: `// sample1.c`
- Output:

  `printf("Enter two +ve integers\n");`
- Input: `scanf("%d%d", &l, &s);`
- Assignment: `rem = l%s;`

- Loop: `while(s){  }`
- Loop Body: `rem = l%s; l = s; s = rem;`

A Variable and Its Memory Location in C

```
int main()
{
    int count ;
    ...............
}
```

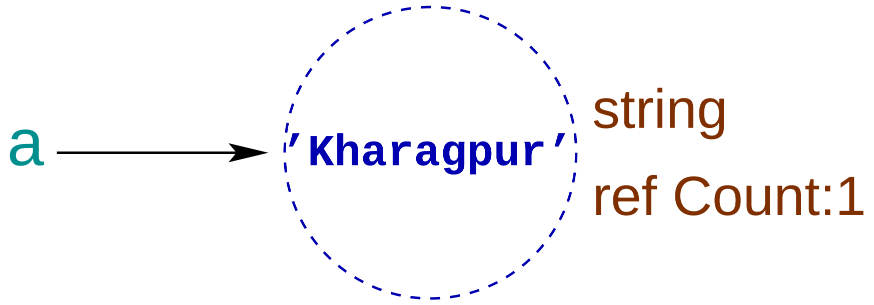

int count;
Content or r−valu
Address or l−value
garbage
type: int
Memory Location for: count

A Variable and Its Value in Python

```
>>> a='Kharagpur'
>>> a
'Kharagpur'
>>> b=a
>>> b
'Kharagpur'
>>> a=5.78
>>> a
5.7800000000000002
```

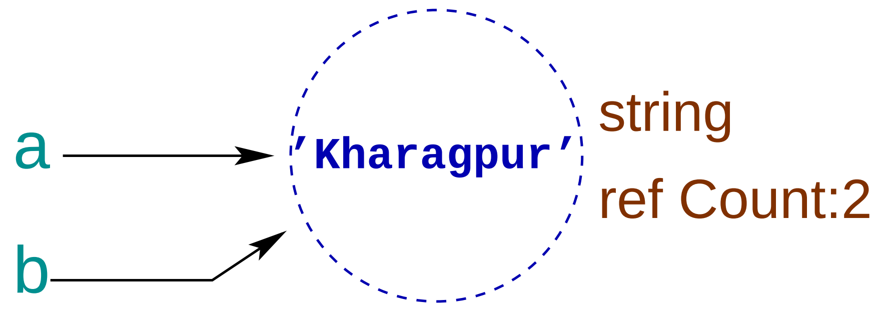
```
>>> a='Kharagpur'
```

- An object `'Kharagpur'` is created along with its type information (string).
- The variable name `a` is created (if it is not already there).
- The name `a` refers to the object `'Kharagpur'`. The reference count of the object is set to one.


a
'Kharagpur'
string
ref Count:1

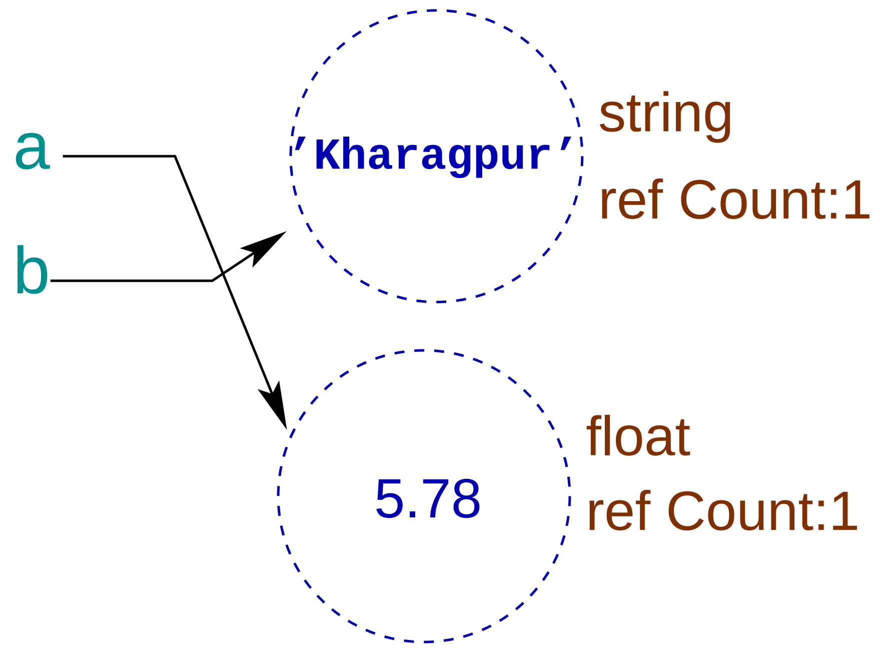
```
>>> b=a
```

- `b` is the second reference to the object `'Kharagpur'`.
- The reference count of the object is incremented by 1.


a
b
'Kharagpur'
string
ref Count:2

```
>>> a=5.78
```

- A new object `5.78` is created along with its type information (floating-point number).
- The name `a` no more refers to the object `'Kharagpur'` and its reference count is reduced by 1.
- Name `a` now refers to the object `5.78` and the reference count of the object is 1.


a
b
'Kharagpur'
string
ref Count:1
5.78
float
ref Count:1

Note

`x = x+5;` in C

is different from

`x = x+5` in Python

`x=x+5` in C

The original content of the location `x` is added to `5` and the value is put in the location of `x`

`x=x+5` in Python

The object referenced by `x` is added to `5`, a new object is created. The name `x` now refers to the new object whose reference count is 1. The reference count of the old object is decremented by 1[a].

[a]If the reference count of an object is zero, its space is reclaimed by the garbage collector.

Some features of Python

Integer, Floating-Point Number, Arithmetic

- Integer and floting-point constants,

  123, -321, -1.23, -1e3, -1.5e-2

- Operators,

  `+, -, *, /, %, **, //`

Examples

```
>>> 13.5+5
18.5
>>> 13.5-5
8.5
>>> 13.5*5
67.5
>>> 13.5/5
2.7000000000000002
>>> 13.5%5
3.5
```

Examples

```
>>> 13.5//5
2.0
>>> 13.5**5
448403.34375
>>>
```

Identifiers and Keywords

- Identifiers,

```
a, A, a1, sum, sum12, min_max1
```

- Keywords,

```
if, while, for, and, ···
```

Variables and Object References

```
>>> a=2
>>> a
2
>>> a=2.5
>>> a
2.5
>>> a="what"
>>> a
'what'
>>> a=[1,2,3,2,1]
>>> a
[1, 2, 3, 2, 1]
```

Variables and Object References

```
>>> a = set([1,2,3,2,1])
>>> a
set([1, 2, 3])
>>> a = 1,2,3,4.5
>>> a
(1, 2, 3, 4.5)
```

Variables and Operators

```
>>> a = 13.5
>>> b = 5
>>> c = a + b
>>> c
18.5
```

Boolean Constants

Relational and Logical Operators

- Boolean Constants,

  `True, False`

- Relational Operators,

  `<, <=, ==, >=, >, !=`

- Logical Operators,

  `and, or, not`

Examples

```
>>> 2 == 3
False
>>> 2 + 3 == 5
True
2 + 3 <= 5
True
2 + 3 > 5
False
```

Change in Control Flow

Depending on data it may be necessary to perform different sets of operations in a program - data dependent execution of sequence of statements (control-flow).

Example

Write a Python Program that reads an integer data from the keyboard. If it is even, it is divided by 2; otherwise 1 is added to it. Print the result.

Program

```
# oddEven.py
n = input("Enter an integer: ")
if n%2 == 0: print "result:", n/2
else: print "result:", n+1
```

Python `if-elif-else` Statement

`if`-statement is used for controlling the execution sequence in a program. The structure or syntax of `if`-statement is as follows.

```
if      $boolean\text{-}expression_1$:
        $statement_1$
elif    $boolean\text{-}expression_2$:
        $statement_2$
⋮       ⋮
elif    $boolean\text{-}expression_{n-1}$:
        $statement_{n-1}$
else:
        $statement_n$
```

Note

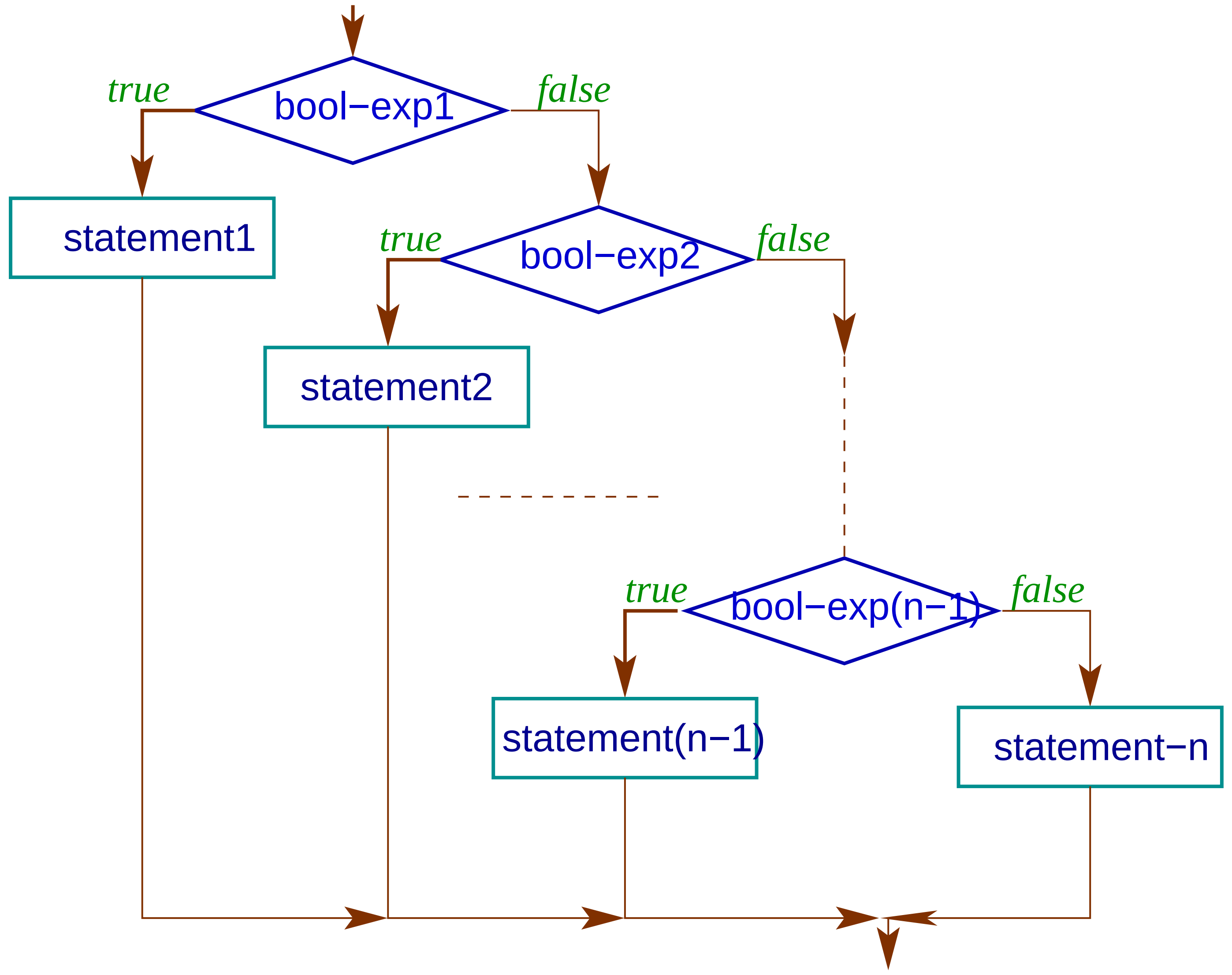
The `elif` and `else` parts are optional.
Every $statement_i$ may be a block of statements with identical indentation.

Indentation

```
# indent1.py : block by indentation
n = input("Enter an integer: ")
if n%2 == 0:
   print n/2
   print n+1
print n
```

Indentation

```
# indent2.py : block by indentation
n = input("Enter an integer: ")
if n%2 == 0:
   print n/2
print n+1
print n
```


true
bool−exp1
false
statement1
true
bool−exp2
false
statement2
true
bool−exp(n−1)
false
statement(n−1)
statement−n

Iteration in Python and C

It is often necessary to execute a sequence of statements (expressions) repeatedly to compute a certain value.

Example

Write a Python program that reads a positive integer n and then reads a set of n integers $a_1, a_2, a_3, \cdots, a_n$. It computes and prints the *arithmetic mean* (AM) of the set of data. The arithmetic mean is defined as,

$$m = \frac{a_1+a_2+a_3+\cdots+a_n}{n}.$$

Program

```
# arithMean.py: Finds arithmetic Mean
n = input('Enter the data count: ')
print 'Enter',n,'data'
sum = input()
i=2
while(i<=n):
    sum = sum + input()
    i = i + 1
print 'AM = ', sum/float(n)
```

Python `while-else` Statement

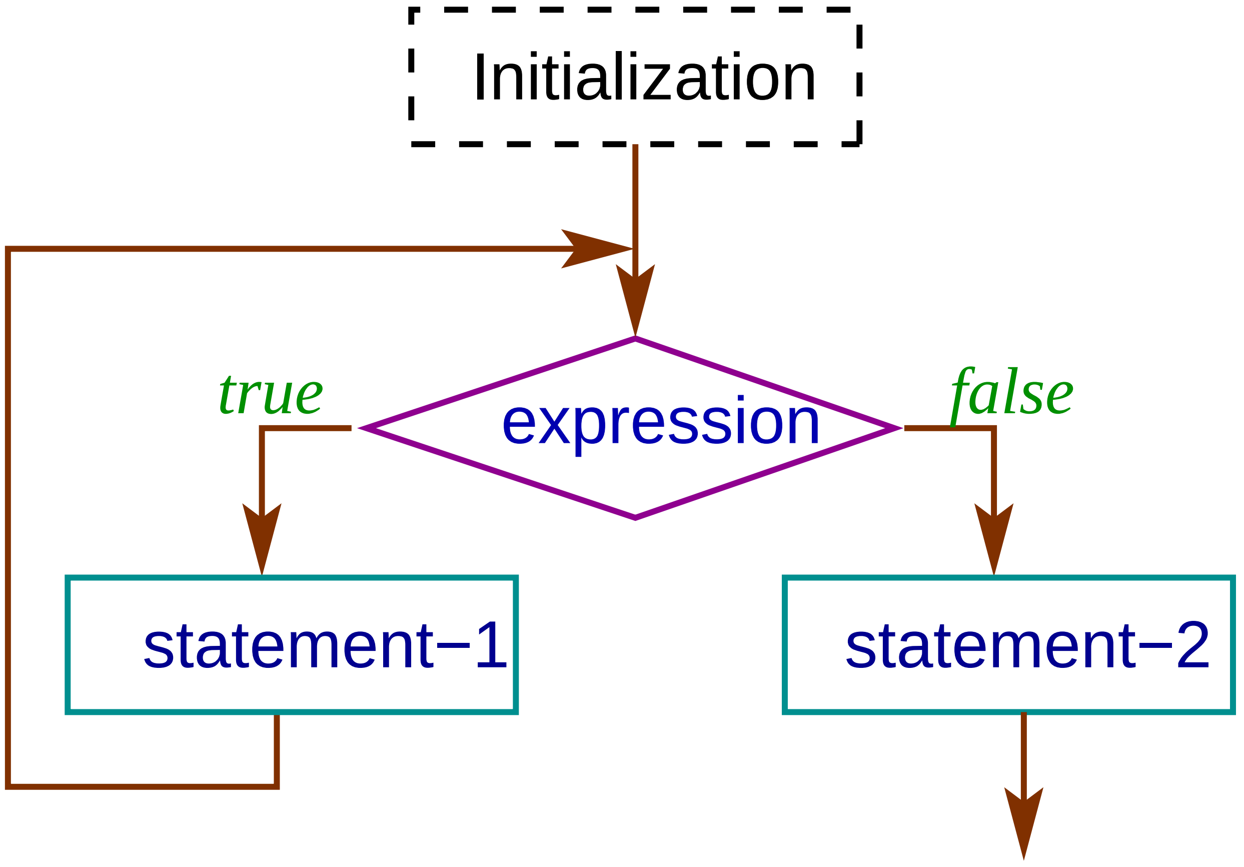
We use `while`-statement to loop through the sequence of statements. The structure or syntax of `while`-statement is as follows.

`while` $\textit{boolean-expression}_1$:

$\quad\quad statement_1$

`else`:

$\quad\quad statement_2$


Initialization
true
expression
false
statement−1
statement−2

Note

The `else` part is optional. Both statements may be a block of statements. The `else` part is executed if the boolean-expression of the `while` is False. So it is always executed after the normal termination of the loop.

Conditional Expression

exp_1 `if` $bool\text{-}exp_1$ `else` exp_2

The value of this expression is exp_1 if the $bool\text{-}exp_1$ is `True`; otherwise the value is exp_2.

Examples

```
>>> a = 5 if 2 <= 3 else 7
>>> a
5
>>> a = 5 if 2 >= 3 else 7
>>> a
7
```

Assignment 1

Write a Python program that reads an integer and prints the sum of its digits at units and tens positions.

Assignment 2

Write a Python program that reads four integers and prints the smallest among them.

Assignment 3

Write a Python program that reads three integers and prints the second largest.

Assignment 4

Write a Python program that reads three positive integers and reports whether they are the lengths of three sides of a triangle. If so, whether they form a Pythagorean triples.

Assignment 5

Write a Python program that reads three positive numbers as the lengths of three sides of a triangle. It computes the area of the triangle using Heron's formula:
$|\Delta| = \sqrt{s(s-a)(s-b)(s-c)}$, where
$s = (a+b+c)/2$. It also computes three altitude of the triangle.

Assignment 6

Write a Python program that reads an integer and counts the number of digits.

Assignment 7

Write a Python program that reads an integer and prints the sum of its digits.

Assignment 8

Write a Python program that reads an integer and prints the count of different decimal digits in it.

`break` and `continue`

A `break`-statement in the body of `while` transfers control outside the loop. The control skips `else` part.
A `continue` statement transfers control to the boolean expression of `while`.