Understanding our first program

JAN 29TH 2014

Our first program

```
n = int(raw_input("Enter a positive integer:"))
while n > 0:
    print n % 2
    n = n/2
```

Last slide on the first line

`n = int(raw_input("Enter a positive integer:"))`

1. `raw_input` prints the prompt, reads a line of the user's input, and returns what is read as a string.

2. This string gets converted to an integer by the function `int`.

3. This integer gets assigned to the variable `n`.

On while-loops

```
Line 1
while boolean expression:
        Line 2
        Line 3
Line 4
```

- while-loops affect the *flow* of the program, i.e., the order in which program statements are executed.

- For the above example the flow of the program is:

Line 1, bool expr (True), Line 2, Line 3, bool expr (True), Line 2, Line 3, bool expr (False), Line 4

Body of while loop

- Lines 2 and 3 form the *body* of the while loop
- Python uses indentation to identify the lines following the while statement that constitute the body of the while loop.

Our first program

```
n = int(raw_input("Enter a positive integer:"))
while n > 0:
    print n % 2
    n = n/2
```

- Suppose `n` has value 35 initially.
- Then the sequence of values that `n` takes on is:

  35, 17, 8, 4, 2, 1, 0.

- When the value of `n` becomes 0, then the boolean expression in the while-statement becomes false and the while-loop ends.

while-loops: Example 2

```
n = int(raw_input("Please type a positive integer: "))

count = 0
while count < n:
    print count
    count = count + 1

print "Done"
```

- What is the output if the user types 10 in response to the prompt?

while-loops: Example 3

```
n = int(raw_input("Please type a positive integer: "))

while n > 0:
    print n
    n = n - 1

print "Done"
```

- What is the output if the user types 10 in response to the prompt?

Boolean expressions

- Python has a type called `bool`

- The constants in this type are `True` and `False`.
  (Not `true` and `false`!)

- The comparison operators:

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

  can be used to construct *boolean expressions*, i.e., expressions that evaluate to `True` or `False`.

Boolean expressions: examples

- Suppose x has the value 10

Expression	Value	Type
x < 10	False	bool
x != 100	True	bool
x <= 10	True	bool
x > -10	True	bool
x >= 11	False	bool

Boolean expressions: more examples

- (12/5) < (12/5.0)
- "100" != 100
- "hello" <= "best"
- int(150.0) == (15 * 10)

Revisiting our program

```
n = int(raw_input("Enter a positive integer:"))
while n > 0:
        print n % 2
        n = n/2
```

- The boolean expression is `True` when n is positive and is `False` when n is less than or equal to 0.
- `n % 2` evaluates to 1 when n is odd and to 0 when n is even.
- `n/2` equals `n/2` when `n` is even and it equals `(n-1)/2` when `n` is odd.
- **Example:** Suppose `n` is initially 25. Then `n` takes on the values (in this order): 25, 12, 6, 3, 1, 0. When `n` becomes 0, the program exits the while-loop.

Improving the output

- How can we put together the bits we generate, in the correct order, to construct the binary equivalent?
- **String concatenation**!

Expression	Value
"0" + "1001"	"01001"
"1" + "1001"	"11001"

Algorithmic idea

- After *i* iterations of the while loop we have generated the right most *i* bits of our answer.

- Call this the *length-i suffix*.

- We want to maintain a string that grows as:

Length-0 suffix → Length-1 suffix → Length-2 suffix →

Example

- Input is 39.

Output	Suffix
1	“”
1	“1”
1	“11”
0	“111”
0	“0111”
1	“00111”
	“100111”

Improved program

```
n = int(raw_input("Enter a positive integer:"))
suffix = “”
while n > 0:
        suffix = str(n % 2) + suffix
        n = n/2
print suffix
```

Further improvement

- Now suppose that we want a more informative output message:

  The binary equivalent of 39 is 100111

- Will this work?

```
n = int(raw_input("Enter a positive integer:"))
suffix = “”
while n > 0:
        suffix = str(n % 2) + suffix
        n = n/2
print “The binary equivalent of “, n, “ is “, suffix
```

Here is what works

```
n = int(raw_input("Enter a positive integer:"))
suffix = ""
originalN = n
while n  > 0:
      suffix = str(n%2) + suffix
      n = n/2
print "The binary equivalent of", originalN, "is",  suffix
```