What would we need to add to introduce *loops* into our language?

Concrete Syntax – what new syntactic forms are in the example below?

Examples (can you think of any without `set!` or `block`?)

```
(let (sum 0)
  (let (i 10)
    (loop
      (if (= i 0) (break sum)
          (block
            (set! sum (+ sum i))
            (set! i (+ i -1)))))))
```

Generated code for examples

Abstract Syntax

Sketch of compiler code changes

If we're OK with 63-bit numbers, can use LSB for *tag*

```
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1010  = 5

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001  = false
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011  = true
```

`cmp <reg>, <val>` — *compute <reg> - <val> and set condition codes (value in <reg> does not change)*

`test <reg>, <val>` — *perform bitwise and on the two values, but don't change <reg>, and set condition codes as appropriate. Useful for mask checking.* `test rax, 1` *will set Z to true if and only if the LSB is 0*

`<label>:` — *set this line as a label for jumping to later*

`jne <label>` — *jump to <label> if Zero is not set (last cmped values not equal)*
`je <label>` — *jump to <label> if Zero is set (last cmped values are equal)*
`jge <label>` — *jump to <label> if Overflow is the same as Sign (which corresponds to >= for last cmp)*
`jle <label>` — *jump to <label> if Zero set or Overflow != Sign (which corresponds to <= for last cmp)*

How to print?

```
<expr> := ... | input | (print <expr>)
```

Consider:

```
(block
  (print 37)
  (print input))
```

Sketch of compiler code changes

```
| Expr::Id(s) if s == "input" => format!("mov rax, rdi")
| Expr::Print(val) =>
```

```
#[no_mangle]
#[export_name = "\x01snek_print"]
fn snek_print(val : i64) {
  if val == 3 { println!("true"); }
  else if val == 1 { println!("false"); }
  else if val % 2 == 0 { println!("{}", val >> 1); }
  else {
    println!("Unknown value: {}", val);
  }
}

#[no_mangle]
#[export_name = "\x01snek_error"]
fn snek_error(code : i64) { … }

fn parse_arg(v : &Vec<String>) -> i64 { … }

fn main() {
    let args: Vec<String> = env::args().collect();
    let input = parse_arg(&args);

    let i : i64 = unsafe { our_code_starts_here(input) };
    print_value(i);
}
```