# Eiffel
# quick lessons



## Class `INTEGER`

- `INTEGER` is just another expanded class
  - Integer literals in the source => creation of an INTEGER object at run-time
  - Eiffel allows assignment of INTEGER to other types

```
n: INTEGER r: REAL d: DOUBLE
..
r := n
d := n
```

  - Division operators:
    - Integer division: `a // b --Returns an INEGER`
    - Integer remainder: `a \\ b --Returns an INEGER`
    - Floating point division: `a / 3.5 --Returns a REAL`



## Floating points

- Supported by classes: REAL, DOUBLE
  - Floating point literal: 9.0, -9.32, 9.5e-3
  - Can be assigned to either `REAL` or `DOUBLE` variables

```
r: REAL d: DOUBLE
r := 9.0
d := -9.32
```

- Floating point `=> INTEGER`

```
d.floor --Returns INTEGER
d.ceiling --INTEGER
d.Rounded --INTEGER
```

  - All these queries are applicable on `REAL` values as well
- `DOUBLE => REAL`

```
d.truncated_to_real --REAL
```



## Characters

- Character literals: Using two single-quotes

```
'a' --The letter a
```

- % is Eiffel's escape character:

```
'%/98/' --The character whose ASCII code is 98
'%N' --New line
'%%' --Percent sign
'%T' --TAB
'%'' --Single quote
'%"' --Double quote
'%H' --Back slash
```

# Class STRING

- Creation

```
"abc" --A string literal
create s.make(6) --Create an empty string,
create s.make_from_string(some_string) --Create and copy
```

- Queries

```
s.count --Length
s.item(5) --5th Character of s
s@5 --5th Character of s
s.substring(2,4) --A new string, copy of 2nd..4th chars.
```

- Commands

```
s.append_string("wxyz") --Append "wxyz" to s
s := s + "wxyz" --Append "wxyz" to s
s.to_lower --s is now all lower case
s.to_upper --all upper
```



# String Conversions

- Value => String
  - Using the `out` query (defined by `ANY`)

```
n: INTEGER s: STRING
..
s := n.out --The same with REAL, BOOLEAN, etc..
```

  - User defined classes can override `out`
- String => Value
  - By invoking a `to_wxyz` query on a `STRING` object

```
n: INTEGER b: BOOLEAN d: DOUBLE s: STRING
..
n := s.to_integer
b := s.to_boolean
d := s.to_double
```



# Console output

- Two equivalent routines
  - `print(s)`
  - `io.put_string(s)`
- io: an attribute of ANY of type STD_FILES
  - `io.put_character(ch)`
  - `io.put_double(d)`
- New line
  - `io.put_new_line`
  - `print("%N")`
- Printing Multiple values
  - Via the string concatenation operator: `+`
  - `print("x=" + x.out + "%N") --Eiffel code`
  - `cout << "x=" << x << endl // C++ code`



# Console input

- Two step process:
  - Accepting a value
  - Assigning into a variable
  - E.g, Reading a string:

```
io.read_line
s := io.last_string -- s is of type STRING
```

- Other types are also supported:
  - `io.read_character, io.read_double, ...`
  - `io.last_character, io.last_double, ...`
  - 0 is returned on “wrong” input

# Command line arguments

- An Eiffel program can receive a command line string
- Class ANY defines two relevant features

```
argument_count --Number of arguments given
argument(n) --Returns the n-th arg. (string)
```

- Name of executable: given by `argument(0)`

- For instance: `C:\eiffel>my_prog a1 a2`

  argument_count = 2
  argument(0) = “my_prog”
  argument(1) = “a1”
  argument(2) = “a2”



# Decisions

- If clause
  - Syntax is very similar to Pascal

```
if a >= low and a <= high then
  print("inside")
  a := (low + high) // 2
elseif a > high then
  print("above")
  a := high + 1
else
  print("below")
  a := low - 1
end
```

  - Comparisons: `=, /=, equal(a,b)`
  - Boolean operators:
    - Standard: `and, or, not`
    - Short circuit: `or else, and then`



# Loops

- A Single loop construct: from..until..loop

```
class SUM
creation {ANY}
   make
feature {}
   make is
      local
         i: INTEGER; s: INTEGER
      do
         from
            i := 0
            s := 0
         until
            i = 10
         loop
            s := s + i
             i := i + 1
         end
    io.put_string("s=" + s.out)
 end
end -- class SUM
```