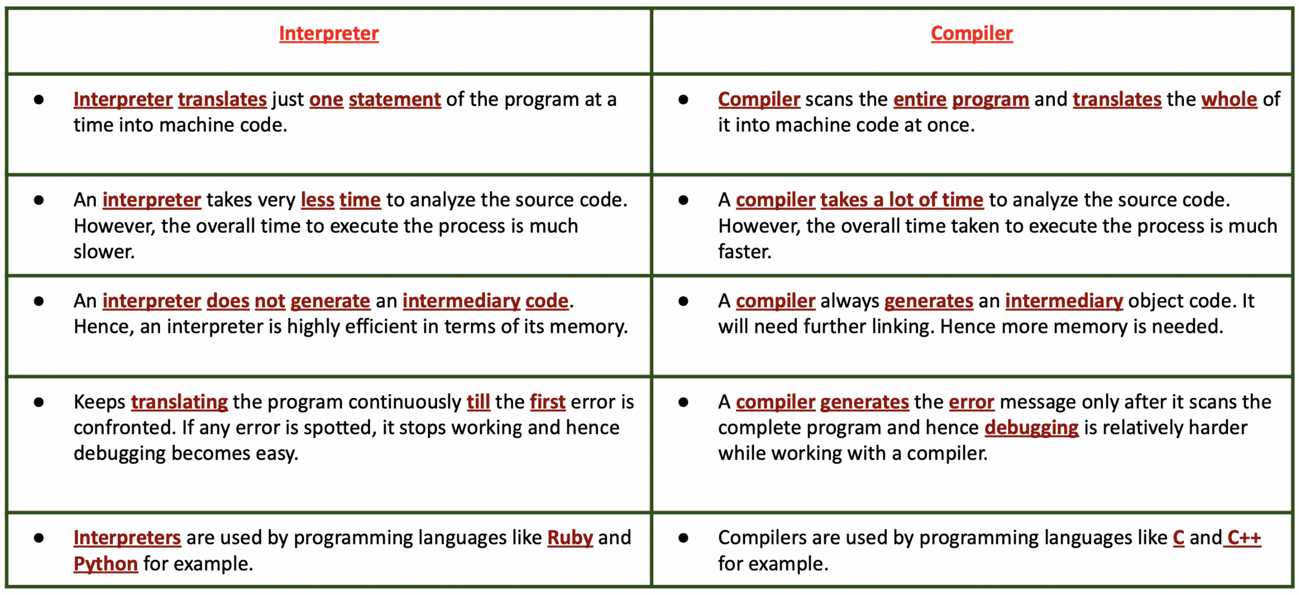
Unit - 1

Interpretors And Compilers

- Converts program to machine readable code.
- first.c -> compiler -> executable -> processor
- first.py -> interpreter -> binary representation -> processor
- Compiler = whole file
- Interpreter = line by line
- **Python** is an interpreted language



Interpreter	Compiler
• Interpreter translates just one statement of the program at a time into machine code.	• Compiler scans the entire program and translates the whole of it into machine code at once.
• An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower.	• A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster.
• An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	• A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.
• Keeps translating the program continuously till the first error is confronted. If any error is spotted, it stops working and hence debugging becomes easy.	• A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.
• Interpreters are used by programming languages like Ruby and Python for example.	• Compilers are used by programming languages like C and C++ for example.

---

pCPS - Process of Computational Problem Solving

1. Analysis
2. Design
3. Implementation
4. Testing

Python Stuff

1. Comments => `#` or `''` or `'''` or `"""`
2. Case sensitive
3. Anything other than special construct should start from column 1
4. You can have multiple statements per line by separating them with `;` but why would you do that

Literals

Are a sequence of one or more characters that stands for itself

Types:

1. **Numeric**: integer, floating point, complex

- Arithmetic overflow and underflow - limitations of floating point
- No such limit for integer type

2. **String** is a literal containing any set of characters enclosed within a pair of quotes.

Identifiers

Are a sequence of one or more characters used to provide a name for a given program element.

- Case sensitive
- Letters, digits and _ (underscores) only
- Should not be a reserved keyword
- Should not start with digits. Must only start with alphabets or underscores.

Variables And Datatypes

1. They point to a memory location
2. A datatype is a set of values and a set of operators that may be applied to those values. For example, integer datatype consists of the set of integers and operators for addition, subtraction etc.
3. **Static and dynamic data declaration:** Static is defined before it's used. Dynamic type doesn't need to be redefined. It dynamically changes data and datatypes as required by the program.
4. If you define variables with the same name, one within a function and one within the global scope, they will not point to the same memory location.

The input() function

1. `input('prompt')`
2. To read a string from standard input
3. Default return type is string
4. You have to convert the return to some other data type as required by your code.

The print() function

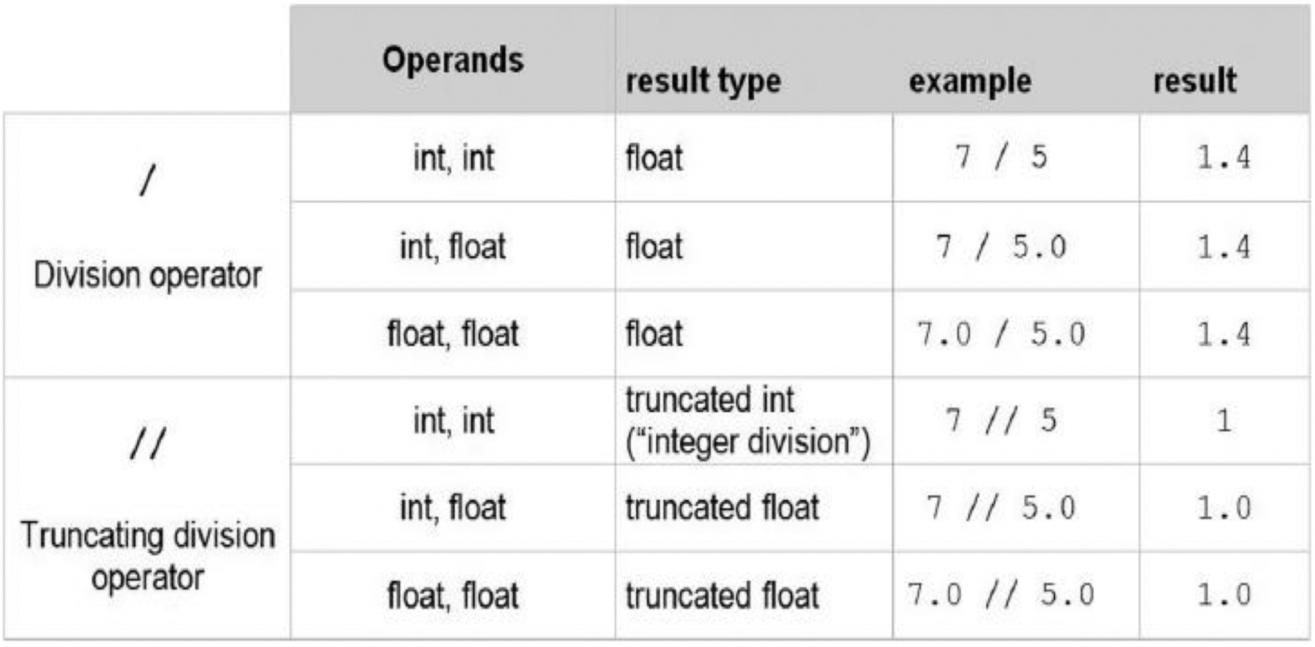
1. `print('Hello World')`
2. Displays on standard output - screen
3. Arguments:

- File: a file like object(stream), defaults to the current `sys.stdout`
- sep: separator
- end: appended after the last value, default = '\n'
- flush: to forcible flush the stream

Operators

are symbol that represent an operation that may be performed on one or more operands.

- Operands are what the operators work on.
- **Unary and Binary operators** work on one, and two operands respectively
- There are relational, binary, integer, membership etc.
- **Important:**

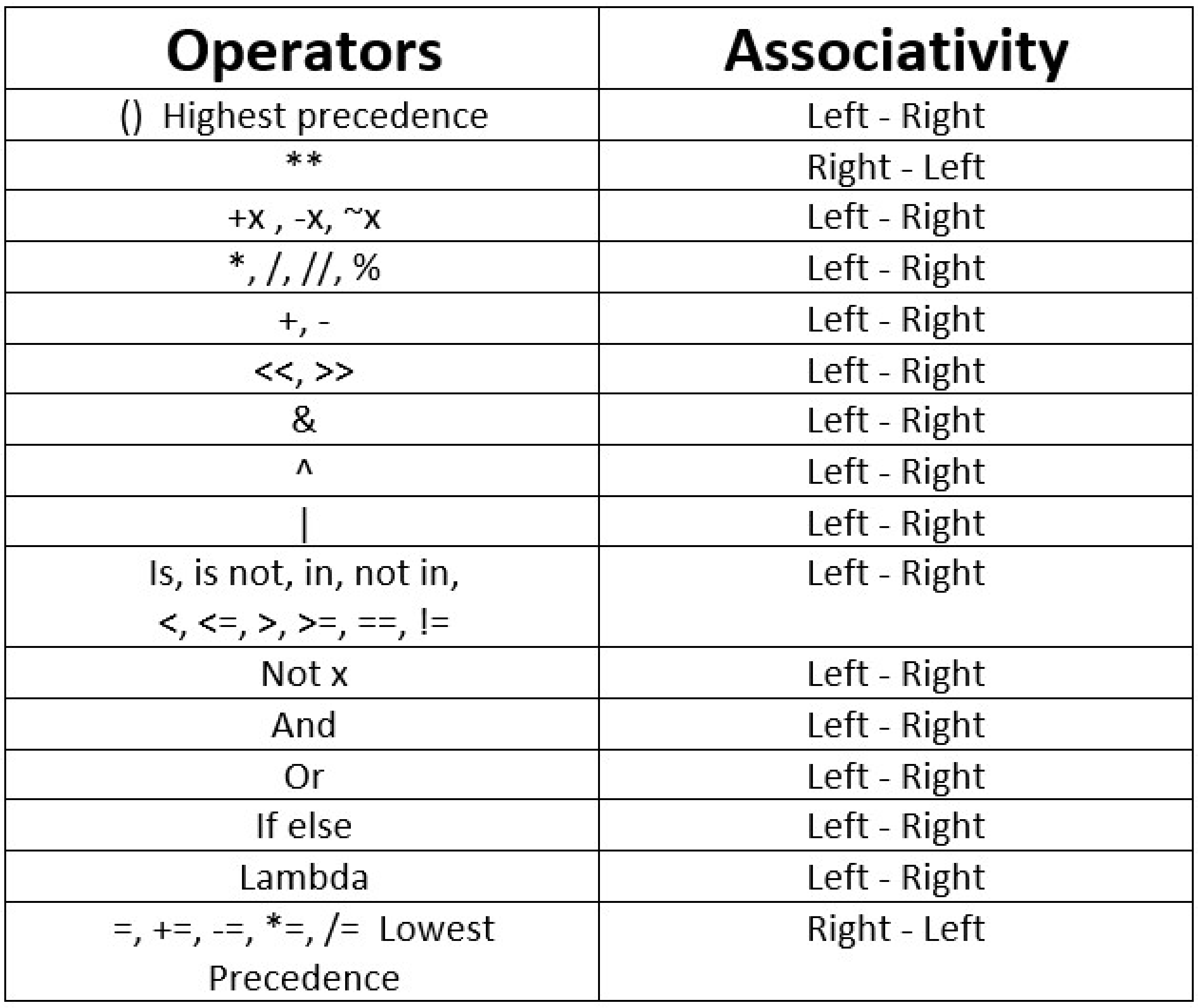


	Operands	result type	example	result
/ Division operator	int, int	float	`7 / 5`	`1.4`
	int, float	float	`7 / 5.0`	`1.4`
	float, float	float	`7.0 / 5.0`	`1.4`
// Truncating division operator	int, int	truncated int ("integer division")	`7 // 5`	`1`
	int, float	truncated float	`7 // 5.0`	`1.0`
	float, float	truncated float	`7.0 // 5.0`	`1.0`

Expressions and Statements

- Expressions are a combination of symbols that evaluate to a value
- Every expression is a statement but it is not true the other way around. Every statement is **not** an expression

Operator Precedence And Associativity



Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

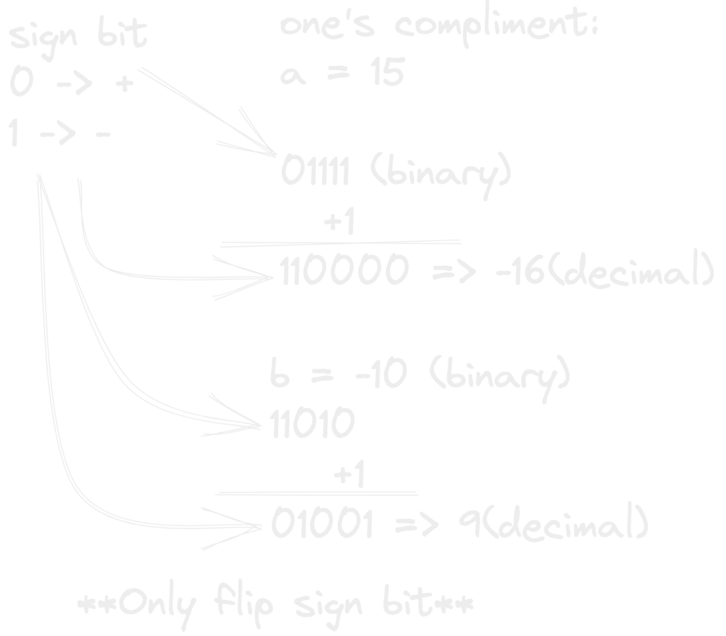
Bitwise Operators

- operations on bits.
Operator	Description
&	And
		Or
^	XOR
<<	Left Shift
>>	Right Shift
~	Compliment

Compliment (kinda weird):

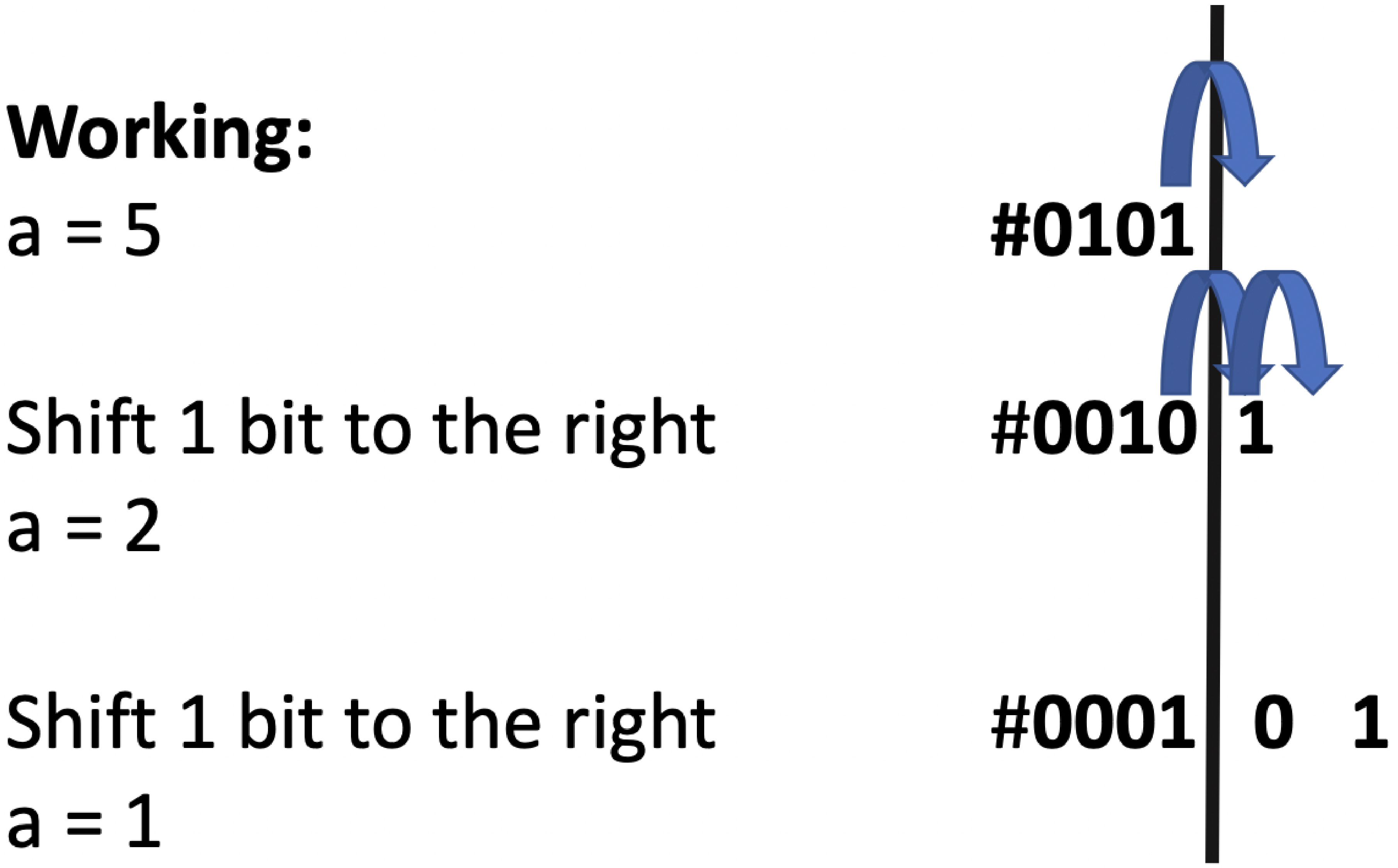
```python
a = 10 = 1010 (Binary)

~a = ~1010
   = -(1010 + 1)
   = -(1011)
   = -11 (Decimal)
```



Right and left shift:

- **Right Shift**: divide by 2 for every right shift
- **Left Shift**: multiply by 2 for every left shift



Working:

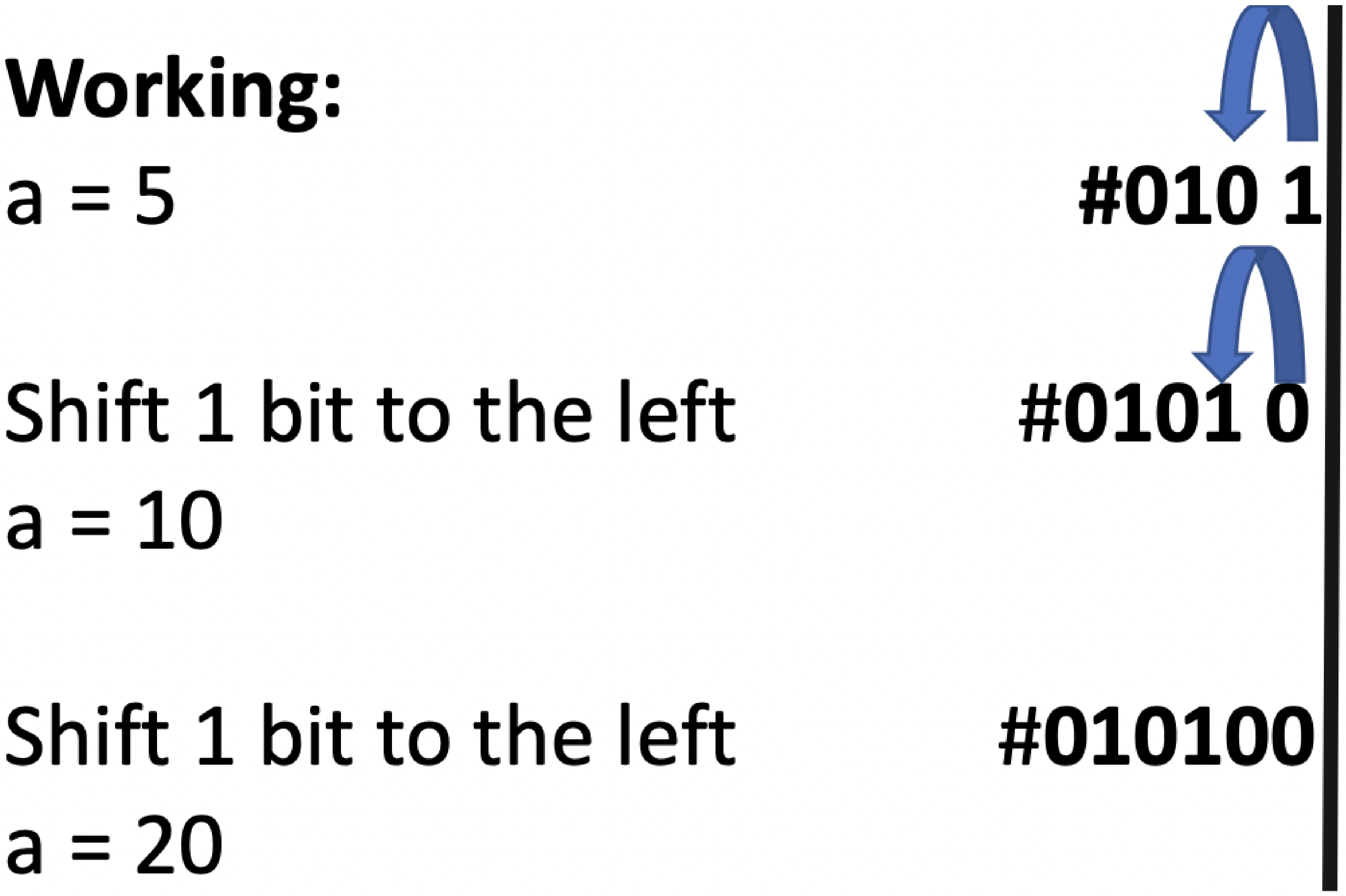
a = 5

Shift 1 bit to the right

a = 2

Shift 1 bit to the right

a = 1



Working:

a = 5 #010 1

Shift 1 bit to the left #0101 0

a = 10

Shift 1 bit to the left #010100

a = 20

You use bitwise operations a lot, if you are doing low level programming on embedded systems.

1. You will have situations where you need to set/clear/toggle just one single bit of a specific register without modifying the other contents. So, you will do a read and do an OR/AND/XOR operation with the appropriate mask for the bit position and write this new value to the register.
2. Usually bitwise operations are faster than doing multiply/divide. So if you need to multiply a variable x by say 9, you will do (x<<3 + x) which would be a few cycles faster than (x*9). If this code is inside an ISR, you will save on response time.
3. Similarly if you want to use an array as a circular queue, it'd be faster(and more elegant) to handle wrap around checks with bit wise operations. (your array size should be a power of 2). Eg: , you can use tail = ((tail & MASK)+ 1) instead of tail = ((tail +1) < size)?tail+1:0, if you want to insert/delete.
4. Also if you want an error flag to hold multiple error codes together, each bit can hold a separate value. You can AND it with each individual error code as a check. This is used in Unix error codes.
5. Also a n-bit bitmap can be a really cool and compact data structure. If you want to allocate a resource pool of size n, we can use a n-bits to represent the current status.

Sample Program:

PYTHON

```python
a, b = 5, 3

print(a&b)

print(a | b)

print(~10)

print(a^b)

print(a<<1)
print(a<<2)

print(a>>1)

print(a>>2)
```

Output:

```
1
7
-11
6
10
20
2
1
```

Relational Operators

- Used to compare 2 values
- Result is of bool type
- The relational Operators Are:

1. <
2. >
3. <=
4. > =
5. ==
6. !=

Logical Operators

- Not
- And
- Or

Membership Operators

- in
- is

How is `is` different from `==`

`is` checks for the memory location, whereas `==` checks the actual value of the operands.

Short Circuit Evaluation

- Logical expression are executed left to right
- Evaluation is stopped as soon as the truth or falsehood is found.

Polymorphism In Operators

Operators that behave differently depending on the operands. For example, `+` can be used for concatenation as well as addition. `*` can be used for numeric multiplication as well as string multiplication.

Control Structures

Control Flow is the order that instructions are executed in a program. A **Control Statement** is a statement that determines control flow of a set of instructions.

There are three fundamental forms of control that programming languages provide,

- Sequential Control
- Selection Control
- Iterative Control

Q. Write a program to read 2 inputs and check the greatest from the 2 numbers.

Q. Find greatest among 3 numbers given by user using if else statements.

Q. Read 5 different subject marks from standard input. Find sum and average of the marks of the student and find the grade awarded to the student.

Q. Find the price of the product after discount if the product price is more than or equal to 5000, the customer gets 20% discount. If it's more than or equal to 3000, customer gets 15% discount. If it's more than 1000, customer gets 10% discount. Print the price of the item after discount.

The `for` loop

- Syntax:

```python
for var in <list|tuple|string|any iterable object>:
    #statements
```

The `range()` function

- built-in function
- usually used in for loop.
- generates an iterable tuple

```python
range(<start>, <stop>, <step>)
```

- it starts from `<start>` to `<stop>-1` by `<step>`

- For example:

```python
range(1,6)
generates
[1,2,3,4,5]

and

range(5)
generates
[0,1,2,3,4]

and

range(1,5,2)
generates
[1,3]
```

- default for `<start>` is `0`
- default for `<step>` is `1`
- `<stop>` is a required argument

The `while` loop

- Syntax

```python
while <condition>:
    #statements
```

Q. Read n range of numbers from standard input, calculate square of each number and print the number with its square.
Q. Write a program to read a sentence displaying each character with its ASCII code using for loop and also by using while loop.
Q. Write a program to print tables from 2 to 10, using for loop and proper formatting.
Q. Write a program to print the fibonacci series.
Q. Write a program to determine if a number is prime or not.
