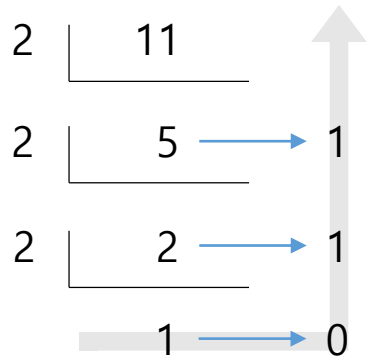
NUMBER SYSTEM

Number System [1]

- Number System Conversion
  - 10 진수를 2 진수로 변환
  - $11_{(10)} \rightarrow 1011_{(2)}$



- 프로그래밍적 사고란?
  - 문제 해결을 위한 논리적 사고를 프로그래밍으로 옮기려는 노력
  - 문제해결을 위한 연산 절차의 모델링
  - 알고리즘 설계

- **진수 변환을 해결하기 위해 고려해야 할 요소들은?**

- 나눗셈을 수행하여야 하는 횟수는 어떻게 획득하는가?
  - 위 예제에는 3번의 나눗셈이 수행됨

- 언제 반복 수행을 중단할 것인가?
  - 반복 횟수만큼 수행 후 중단
  - 위 예제에는 몫이 < 2 인 경우 중단

- 최종 출력/해를 어떻게 구성할 것인가?
  - 나머지를 역순으로 연결
  - 현재 획득한 나머지를 이전 나머지의 앞에 위치
  - 마지막으로 몫을 연결

Number System [2]

- Lab.01
  - 먼저 $11_{(10)}$ → $1011_{(2)}$ 만을 고려한다.
  - 10 진수 0 의 입력은 현재 고려하지 않는다.
  - 파일명: two_system.py

```
BASE = 2 # 2진수
num_dec = 11 # 입력되는 10 진수
print(f"Decimal number = {num_dec} " ) # 입력 10진수 출력

Num_bin = “ ” # 출력을 위한 공백 문자열을 준비

# 총 3번의 나눗셈으로 구성
# 1번째 나눗셈
Num_dec, r = num_dec // BASE, num_dec % BASE # 2로 나눈 몫과 나머지
Num_bin = str(r) + num_bin # 출력 문자열 구성

# 2번째 나눗셈
num_dec, r = num_dec // BASE, num_dec % BASE
num_bin = str(r) + num_bin

# 3번째 나눗셈
num_dec, r = num_dec // BASE, num_dec % BASE
num_bin = str(r) + num_bin

# 마지막 나눗셈의 몫을 연결
Num_bin = str(num_dec) + num_bin #

# 최종 계산된 2진수를 출력
print(f"Binary number = {num_bin}")
```

two_system.py

- 고려 사항
  - *동일 기능의 코드가 중복 나열되어 있음*
    - 3번의 나눗셈이 반복 적용됨
  - *다른 문제에 대하여 중복 코드를 계속 반복하여 해결해야 하는가?*
    - 즉, $11_{(10)}$인이 아닌 다른 문제에 $54_{(10)}$과 같이 나눗셈의 횟수가 다른 문제
    - $54_{(10)}$는 5 번의 나눗셈이 필요
  - *나눗셈의 반복횟수를 결정할 수 있다면?*

Number System (3)

- 순환문
  - 추후 배울 내용이지만 잠시 소개를 하자면...
  - for-loop
  - while-loop

- while-loop
  - https://docs.python.org/3.7/reference/compound_stmts.html#the-while-statement

8.2. The `while` statement

The `while` statement is used for repeated execution as long as an expression is true:

```
while_stmt ::=  "while" expression ":" suite
                ["else" ":" suite]
```

This repeatedly tests the expression and, if it is true, executes the first suite; if the expression is false (which may be the first time it is tested) the suite of the `else` clause, if present, is executed and the loop terminates.

A `break` statement executed in the first suite terminates the loop without executing the `else` clause's suite. A `continue` statement executed in the first suite skips the rest of the suite and goes back to testing the expression.

Number System (4)

- Example
  - while-loop
  - 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 3 + 1

```
cnt_iter = 10

sum_ = 0

while cnt_iter > 0:
    sum_ += cnt_iter
    cnt_iter -= 1

print(f"sum = {sum_}, cnt_iter = {cnt_iter}") # output sum = 55, cnt_iter = 0
```

clause

header

suite

Number System [5]

- Lab.02
  - 파일명: two_system_iter.py

```
BASE = 2 # 2진수
num_dec = 11 # 입력되는 10 진수
print(f"Decimal number = {num_dec}") # 입력 10진수 출력

num_bin = "" # 출력을 위한 공백 문자열을 준비

# 총 3번의 나눗셈
# 1번째 나눗셈
Num_dec, r = num_dec // BASE, num_dec % BASE # 2로 나눈 몫과 나머지
Num_bin = str(r) + num_bin # 출력 문자열 구성

# 2번째 나눗셈
num_dec, r = num_dec // BASE, num_dec % BASE
num_bin = str(r) + num_bin

# 3번째 나눗셈
num_dec, r = num_dec // BASE, num_dec % BASE
num_bin = str(r) + num_bin

# 마지막 나눗셈의 몫을 연결
Num_bin = str(num_dec) + num_bin #

# 최종 계산된 2진수를 출력
print(f"Binary number = {num_bin}")
```

two_system.py

- 고려 사항
  - *나눗셈의 반복횟수를 결정할 수 있다면?*
    - $y = \log_2(x)$
    - https://docs.python.org/3.7/library/math.html#math.log
    - import math 필요
  - while-loop 를 사용하여 단순화한다.
    - *결정된 반복 횟수 y 만큼 순환한다.*

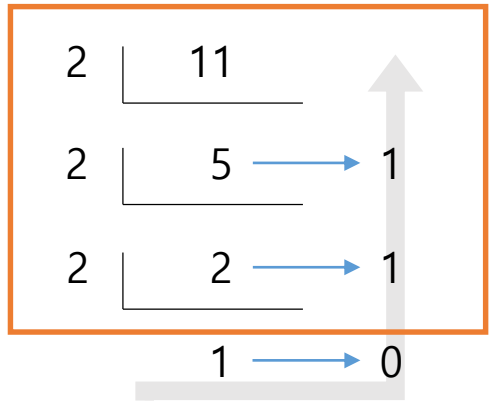
Number System [6]

- Lab.02-01
  - 10 진수 0 의 입력은 현재 고려하지 않는다.
  - 파일명: two_system_iter_00.py

```
import math

BASE = 2
num_dec = 54
print(f"Decimal number = {num_dec}")

cnt_iter = int(math.log(num_dec,2))
num_bin = ""

while cnt_iter > 0:
    num_dec, r = num_dec // BASE, num_dec % BASE
    num_bin = str(r) + num_bin
    cnt_iter -= 1

num_bin = str(num_dec) + num_bin

print(f"Binary number = {num_bin}")
```

two_system_iter_00.py

- 고려 사항
  - 다양한 10 진수의 값을 입력에도 결과를 산출
    - $11_{(10)}$ → $1011_{(2)}$ 출력
    - $54_{(10)}$ → $110110_{(2)}$ 출력
  - 순환문의 조건에 대하여 고려
    - 반복 횟수의 결정하는 연산이 필요한가?
    - `cnt_iter = int(math.log(num_dec,2))`
  - 불필요한 연산을 줄이려는 노력이 필요
    - 반복 횟수를 결정하는 연산을 사용하지 않는다면?



- 어느 조건에서 나눗셈을 수행하는가?

Number System [7]

- Lab.03
  - 10 진수 0 의 입력을 고려한다.
  - 파일명: two_system_iter_01.py

```
BASE = 2
num_dec = 13
print(f"Decimal number = {num_dec}")

num_bin = ""

while num_dec >= BASE:
    num_dec, r = num_dec // BASE, num_dec % BASE
    num_bin = str(r) + num_bin

num_bin = str(num_dec) + num_bin

print(f"Binary number = {num_bin}")
```

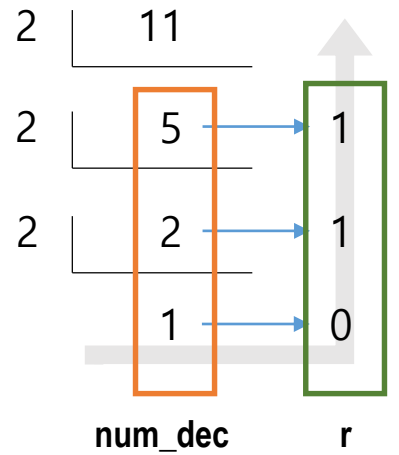
two_system_iter_01.py

- 고려 사항
  - 반복 횟수를 결정하는 연산을 제거

```
cnt_iter = int(math.log(num_dec,2))
```

  - while header 의 조건문을 수정한다.

```
while num_dec >= BASE:
```

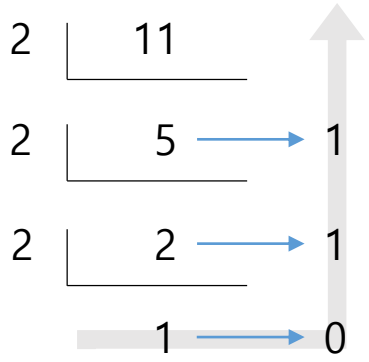


    - 2보다 같거나 큰 경우에만 나눗셈 연산을 수행한다.

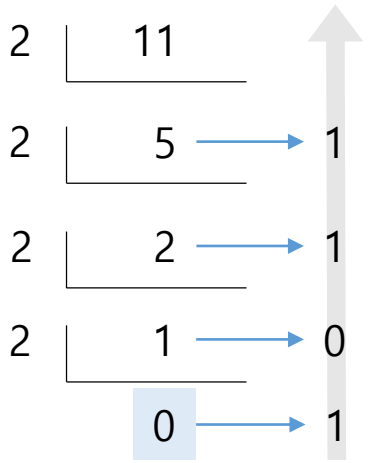
  - `num_bin = str(num_dec) + num_bin` 에 대하여 고려해보자.

Number System [8]

- Lab.04
  - (1) 과 (2) 의 논리 모델을 고려해 보면?
  - (1)



  - (2)



- 고려 사항
  - 유니폼(uniform)화 된 모델이란?
    - 하나의 작업이 분리된 절차들로 구성되어있는 것이 아닌 동일한 성격의 절차들로 로 uniform 화 되어 있는 모델
    - 정해진 기준에 의해 정해진 동일한 코드의 형태
  - (2) 번의 형태가 보다 더 유니폼화된 모델
  - `num_bin = str(num_dec) + num_bin` 에 대하여 제거를 고려
  - 순환문의 중단 조건을 고려하자.
  - 좀 더 완성도를 높일 수는 없는가?
    - 설계된 논리 모델을 수행하기 이전에 필터링 되는 조건은?
    - BASE 보다 작은 10 진수가 입력되는 경우
  - 논리 모델에서의 필터링 요소란?
    - 설계된 모델의 정상적인 수행을 방해하는 요소
    - 설계된 모델로의 진입이 불필요한 요소
  - 필터링을 위한 조건문을 사용
    - if-else

Number System [9]

- 조건문
  - 추후 배울 내용이지만 잠시 소개를 하자면...
  - https://docs.python.org/3.7/reference/compound_stmts.html#the-if-statement

8.1. The if statement

The `if` statement is used for conditional execution:

```
if_stmt ::=  "if" expression ":" suite
             ("elif" expression ":" suite)*
             ["else" ":" suite]
```

It selects exactly one of the suites by evaluating the expressions one by one until one is found to be true (see section Boolean operations for the definition of true and false); then that suite is executed (and no other part of the `if` statement is executed or evaluated). If all expressions are false, the suite of the `else` clause, if present, is executed.

- Example

```
num = 3

if num % 2:
    print(f"{num} = odd number")
else:
    print(f"{num} = even number")
```

Number System (10)

- Lab.04-01
  - 파일명: two_system_iter_02.py

```
BASE = 2
num_dec = 51
print(f"Decimal number = {num_dec}")

num_bin = ""

if num_dec < BASE:
    num_bin = num_dec
else:
    while num_dec > 0:
        num_dec, r = num_dec // BASE, num_dec % BASE
        num_bin = str(r) + num_bin

print(f"Binary number = {num_bin}")
```

two_system_iter_02.py

- 고려 사항
  - 설계된 논리 모델을 수행하기 이전에 필터링 되는 조건은?
    - BASE 보다 작은 10 진수가 입력되는 경우
    - 논리 모델로 진입할 필요가 없다.
  - 조건문을 사용한 필터링
  - 8진수를 시험해본다.
    - BASE 를 8 로 변경하여 시험

```
BASE = 8
…
print(f"Otal number = {num_bin}")
```

  - *16 진수는?*
    - *조건문을 잘 활용하면16 진수도 해결 가능*

Number System (11) – HW

- HW
  - 파일명: hex_학번.py

- 구현 시 고려사항
  - 아래를 참고하여 반드시 조건문을 사용할 것!

Dec'	Hex
0	0
1	1
…	…
10	A
11	B
12	C
13	D
14	E
15	F

  - 이전 코드 two_system_iter_02.py 를 응용할 것!
  - 필터링 고려할 것!

Number System [12]

- Python document 활용
  - https://docs.python.org/3.7/library/functions.html#bin
  - 개발 환경의 Python 버전을 선택하여 확인

bin(*x*)

Convert an integer number to a binary string prefixed with "0b". The result is a valid Python expression. If *x* is not a Python `int` object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> bin(3)
'0b11'
>>> bin(-10)
'-0b1010'
```

If prefix "0b" is desired or not, you can use either of the following ways.

```
>>> format(14, '#b'), format(14, 'b')
('0b1110', '1110')
>>> f'{14:#b}', f'{14:b}'
('0b1110', '1110')
```

See also `format()` for more information.

Number System [1]

- Binary arithmetic operation

```
x_01 = 0b1010
x_02 = 0b0011
y = x_01 + x_02

print(bin(y)) # Ouput 0b1101
print(y) # Ouput 13
```

```
x_01 = 0xA
x_02 = 0x3
y = x_01 % x_02

print(bin(y)) # Ouput 0b1
print(y) # Ouput 1
```

```
x_01 = 0b1010
x_02 = 0b0011
y = x_01 - x_02

print(bin(y)) # Ouput 0b111
print(y) # Ouput 7
```

```
x_01 = 0xA
x_02 = 0x3
y = x_01 // x_02

print(bin(y)) # Ouput 0b11
print(y) # Ouput 3
```

```
x_01 = 0o12
x_02 = 0o3
y = x_01 * x_02

print(bin(y)) # Ouput 0b11110
print(y) # Ouput 30
```

```
x_01 = 0b1010
x_02 = 0b0011
y = x_01 / x_02

print(bin(y)) # Error
print(y) # Ouput 3.333333333333335
```

TypeError:
'float' object cannot be interpreted as an integer

Number System [13]

- Bitwise logical operation

```
x_01 = 0b1010
x_02 = 0b0011
y = x_01 & x_02

print(bin(y))  # Ouput 0b10
print(y)  # Ouput 23
```

```
x_01 = 0o12
x_02 = 0o3
y = x_01 ¦ x_02

print(bin(y)) # Ouput 0b1011
print(y) # Ouput 11
```

```
x_01 = 0xA
x_02 = 0x3
y = x_01 ^ x_02

print(bin(y))  # Ouput 0b1001
print(y)  # Ouput 9
```

```
x = 0b1010
y = ~x
print(bin(y))  # Ouput -0b1011
```

- ob0101 로 표현되지 않는다.
- 2' complement 에 대하여 python 자체적으로 사용되는 표현식이다.
- 보수에서 설명한다.

```
x = 0x3
y = x << 2

print(bin(y))  # Ouput 0b1100
print(y)  # Ouput 12
```

```
x = 0o14
y = x >> 2

print(bin(y))  # Ouput 0b11
print(y)  # Ouput 3
```

Number System [14]

- 보수 (Complement)
  - 보충해주는 수
  - N's complement = (N-1)'s complement + 1

- 10진수에서의 9의 보수
  - $697_{(10)}$

	6	9	7
+			
	9	9	9

  - **어떤 수**를 더해야(보충해 주어야) 각 자리수에 대하여 9 를 만족하는 가?

	6	9	7
+	**3**	**0**	**2**
	9	9	9

  - 999 – 302 와 동일
  - $697_{(10)}$ 에 대한 9의 보수는 $302_{(10)}$

- 10진수에서의 10의 보수
  - $748_{(10)}$

	6	9	7
+			
	1	0	0	0

  - **어떤 수**를 더해야(보충해 주어야) 각 자리수에 대하여 10 를 만족하는 가? 자리수의 올림수를 고려하여 10을 만족하여야 한다.

	6	9	7
+	**3**	**0**	**3**
	1	0	0	0

  - 즉 9의 보수에 1을 한 것과 동일
  - 10^3 – 697 과 동일

	3	0	2
+			**1**
	3	**0**	**3**

Number System [16]

- 보수 (Complement)

  N's complement = (N-1)'s complement + 1

- 2진수의 보수
  - 1의 보수
  - 2의 보수 = 1의 보수 + 1

- $0101_{(2)}$ 에서의 1의 보수

0	1	0	1

  - 각 자리수의 값을 반전(0은 1, 1은 0으로 변환)
    - 1111 과의 XOR 연산을 수행

	0	1	0	1
XOR	1	1	1	1
	1	**0**	**1**	**0**

- $0101_{(2)}$ 에서의 2의 보수
  - 1의 보수 + 1

	1	0	1	0
+	0	0	0	1
	1	**0**	**1**	**1**

Number System [17]

- Python 의 ~ operator
  - 저장 형태는 2 의 보수의 값으로 저장되며
  - 표현방식은 Python 자체의 표현 방식을 따른다.

```
x = 0b1010
y = ~x
print(bin(y))  # Ouput -0b1011
```

- 수행 결과는 기대한 0b0101 아니다.
  - 먼저 수행 결과 –b1011 은 2의 보수의 저장 방식을 따르므로
  - 2' 보수 = 1' 보수 + 1

	1	0	1	1
-	0	0	0	1
	1	0	1	0
XOR	1	1	1	1
	0	1	0	1