Python(+System Engineer 개요)

<python 함수의 구조>

```
def 함수명(매개변수):
    <수행할 문장1>
    <수행할 문장2>
    ...
```

```
def add(a, b):
    return a+b

print(add(5,3))
```

→ 5+3 =8 8이 출력이 됨

<나머지 응용>

```
def minus(a, b):
    return a-b

print(minus(5,3))

def multiply(a, b):
    return a*b

print(multiply(5,3))

def divide(a, b):
    return a/b

print(divide(6,3))
```

```
def add(a, b):
    return a+b
a = 3
b = 4
c = add(a, b)
print(c)
```

<응용>

```
def minus(a, b):
```

```
    return a-b

a = 7
b = 4
c = minus(a, b)
print(c)

def multiply(a, b):
    return a*b

a = 3
b = 4
c = multiply(a, b)
print(c)

def divide(a, b):
    return a/b

a = 8
b = 4
c = divide(a, b)
print(c)
```

<입력값이 없는 함수>

```
def say():
    return 'Hi'

a = say()
print(a)
```

say를 호출해서 Hi라는 값을 Return시켜서 Hi를 출력시킴

<결괏값이 없는 함수>

```
def add(a, b):
    print("%d, %d의 합은 %d입니다." % (a, b, a+b))

add(5,3)
```

→ Return값이 없어서 처리 결과만 돌아옴.

<입력값도 결괏값도 없는 함수>

```
def say():
```

```
    print('Hi')
```

<함수 호출할때 매개변수 지정하기>

```
def add(a,b):
    result = a+b
    return result

result = add(a=5,b=3)
print(result)
```

<함수의 입력값이 몇 개가 될지 모를때>

```
def 함수이름(*매개변수):
    <수행할 문장>
    ...

def add_many(*args):
    result = 0
    for i in args:
        result = result + i
    return result
```

<argument와 parameter>

argument : 인수, 인자 ↔ parameter : 매개변수

Linux Shell에서 명령어를 타이핑하고 호출할때 사용함 ex) ls -l test.txt 에서 echo $0하면 ls가 출력되게 됨. (-l : $1, test.txt : $2)

```
args = [1,2,3,4,5]
for i in args:
    print(i)

result = 0
args = [1,2,3,4,5]
for i in args:
    result = result + i
print(result)
```

※ <시스템 용어> - SE(시스템 엔지니어) 직업 설명

Docker

Kubernetes(쿠베네티스)

Ansible

Terraform

...
YAML

● def add_many(*args) 활용

```
def add_many(*args):
        result = 0
        for i in args:
             result = result + i
        return result
result = add_many(1,2,3)
print(result)
```

※ args는 인수를 뜻하는 영어 단어 arguments의 약자이며 관례적으로 자주 사용한다.

```
result = add_mul('mul', 1,2,3,4,5)
print(result)
```

```
def add_and_mul(a,b):
        return a+b, a*b
```

<매개변수 기본값 설정하기>

```
def say_myself(name, old, man=True): → 매개변수 마지막에 기본값을 적용해야함.
      print("나의 이름은 %s 입니다." % name)
      print("나이는 %d살입니다." % old)
      if man:
           print("남자입니다.")
      else:
           print("여자입니다.")

say_myself('홍길동', 23)
```

<함수 안에서 함수 밖의 변수를 변경하기>

```
a = 1
def vartest(a):
      a = a +1
      print("함수안의 a의 값 %d" % a)

vartest(a)
print("함수밖의 a의 값 %d" % a)
```

■ input 함수

a=input() -> 키보드로 입력하는 값을 문자처리(string type)함.
1234
a
'1234'
type(a)
<class 'str'>

■print 함수
print("life" "is" "too short")
lifeistoo short
→"life" + "is" + "too short"와 같음
print("life", "is", "too short")
life is too short

■파일 열기, 쓰기, 닫기, 읽기
```
# 파일 열기
f = open('python/새파일.txt', 'w')

# 파일 쓰기
for i in range(1, 11):
    data = "%d번째 줄입니다. \ n" % i
    f.write(data)

# 파일 닫기
f.close()

# 파일 쓰기 추가(append)
for i in range(11, 21):
    data = "%d번째 줄입니다. \ n" % i
    f.write(data)
# 파일 읽기모드로 열기
f = open("C:/doit/새파일.txt", 'r')
line = f.readline()
print(line)
f.close()

<개행문자 없애기>
lines = f.readlines()
for line in lines:
    line = line.strip()
    print(line)
```

<read 함수 이용해서 파일 읽기>

```
data = f.read()
print(data)
```

```
# with문과 함께 사용하기
f = open("새파일.txt", 'w')
f.write("Life is too short, you need python")
f.close()
```

```
with open("foo.txt", 'w') as f:
    f.write("Life is too short, you need python")
```

```
# Module로 매개변수 주기
import sys → improt 붙여오는것
```

```
args = sys.argv[1:]
for i in args:
    print(i)
```

```
[root@localhost ~]# echo "Hi"
Hi
[root@localhost ~]# a="Hello World"
[root@localhost ~]# echo $a
Hello World
[root@localhost ~]#
```

```
[root@localhost ~]# cat > test.txt
Hello World
^C
[root@localhost ~]# cat test.txt
Hello World
```

```
[root@localhost ~]# python -c 'print("A" *5)'
AAAAA
[root@localhost ~]# python -c 'print "A" *5'
AAAAA
```

컴파일링 없이 바로 실행함.

```
C:₩git_repo₩python>python sys1.py aaa bbb ccc
import sys
```

```
args = sys.argv[1:]
for i in args:
    print(i.upper())
```