Recursion

↳ when a function calls itself

until a specific condition is met

There is a function

```
f(){

    f()

}
```

(understanding the theory)

Part 1

↳ when a function calls itself

Example

```
f(){            → f(){          → f(){          → f(){
  print(1);        print(1);       print(1);       print(1);
  f();             f();            f();            f();
}               }               }               }
```

```
main(){
  f();
}
```

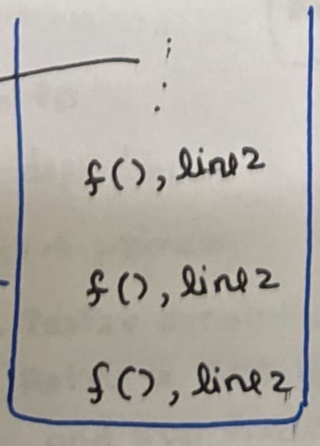
(why this function keeps on continuously calling itself) → cause there is no final condition that is met

(Internally what happens)

we call a function as complete only when it executes till the last line

(you have a stack)

(goes on)



leads to Segmentation fault → Stack overflow

O/P

1
1
⋮
1

→ This has a specific memory and can't keep on stack/ compile lot of memory calls

Segmentation fault

↳ when there is numerous function calls which are waiting due to Recursion → stack overflow

(Problem of Infinite Recursion)

Part-2

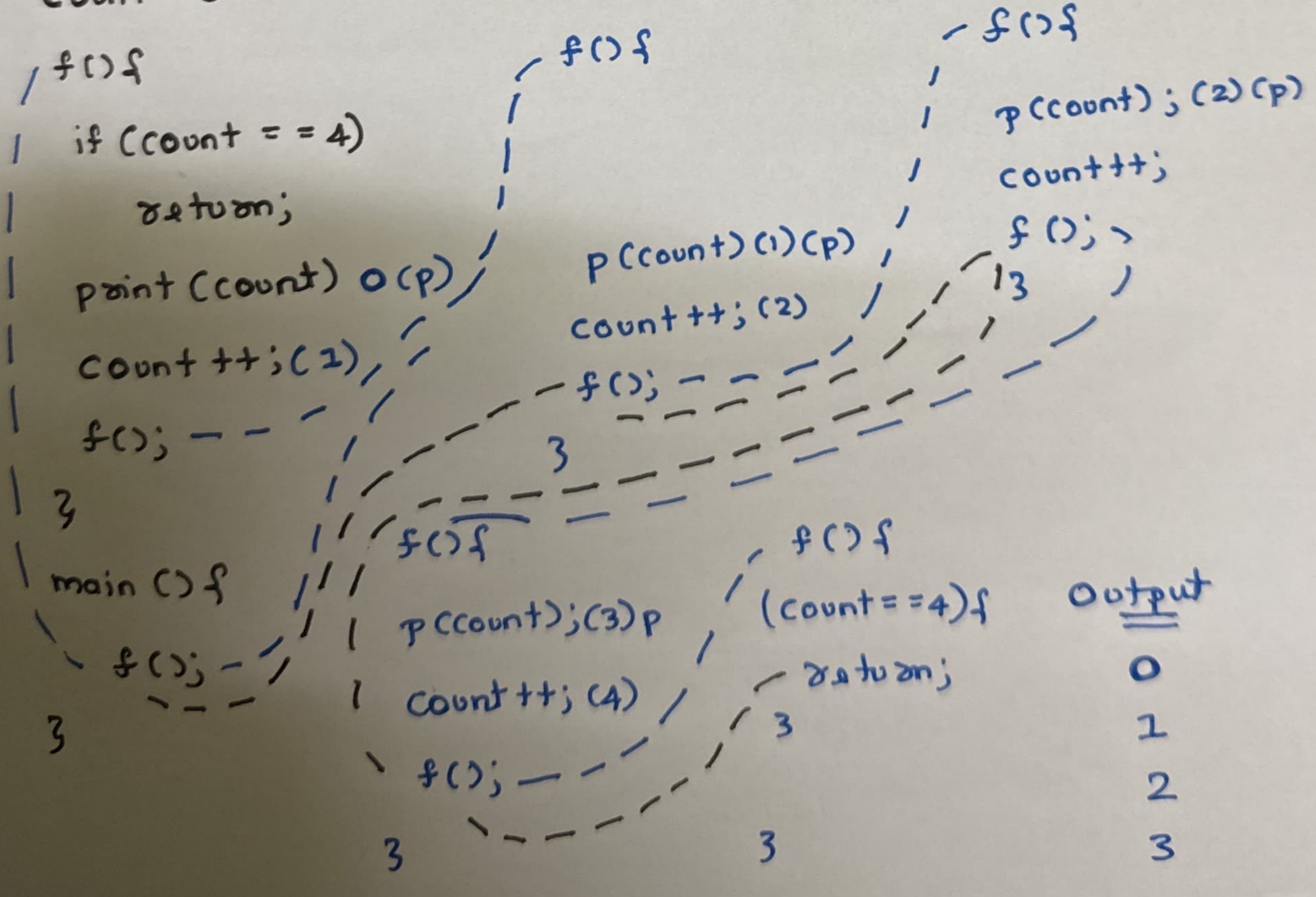
Unless and until a specific condition is met

(Base Condition)

↳ condition you use to stop the recursion

(Look into the flow properly)

Count = 0



```
f(){
  if (count == 4)
     return;
  print (count) 0 (p)
  count ++; (1)
  f();
}
main (){
  f();
}
```

```
f(){
  p (count) (1) (p)
  count ++; (2)
  f();
}
```

```
f(){
  p (count); (2) (p)
  count ++;
  f();
}
```

```
f(){
  p (count); (3) p
  count ++; (4)
  f();
}
```

```
f(){
  (count==4){
  return;
}
}
```

Output

```
0
1
2
3
```