# 01158 - ALGORITMA DAN STRUKTUR DATA 2

# STACK

# 04

OLEH : ANDRI HERYANDI, M.T.

# MATERI

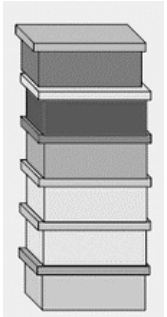
## 01158 - ALGORITMA DAN STRUKTUR DATA 2

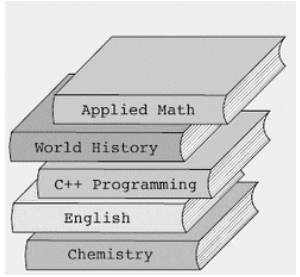
- Definisi Stack
- Contoh Penerapan Stack
- Operasi-Operasi Pada Stack
- Contoh Implementasi Stack

# DEFINISI STACK



- Dalam Bahasa Indonesia disebut sebagai **Tumpukan**
- Stack adalah sebuah daftar elemen data dimana penambahan dan pengambilan/penghapusan data hanya dilakukan pada posisi akhir.
- Menerapkan kaidah **Last In First out (LIFO)**
- Dalam stack, posisi akhir disebut sebagai **Top**.
- Apa pun implementasi yang dilakukan, 2 aturan berikut harus benar-benar dipatuhi.
  - Push : Penambahan data hanya akan selalu dilakukan pada posisi Top.
  - Pop : Pengambilan data hanya akan dilakukan pada posisi Top.

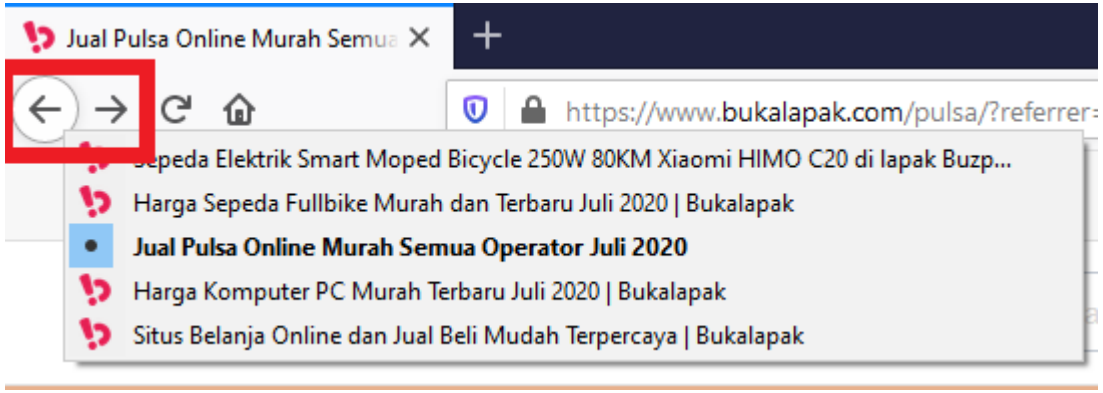



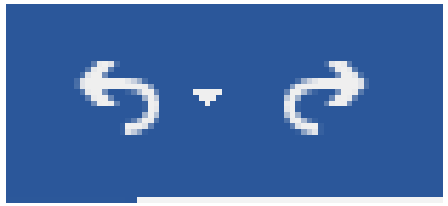


# CONTOH PENERAPAN STACK



- Contoh penerapan stack di dunia kehidupan sehari-hari
  - Tumpukan piring di restoran
- Contoh penerapan stack di dunia komputer
  - Tombol Next dan Back di web browser
  - Undo – Redo
  - Penghitungan rumus matematika (dengan memperhatikan urutan prioritas operator)
  - Penyelesaian pencarian rute labirin

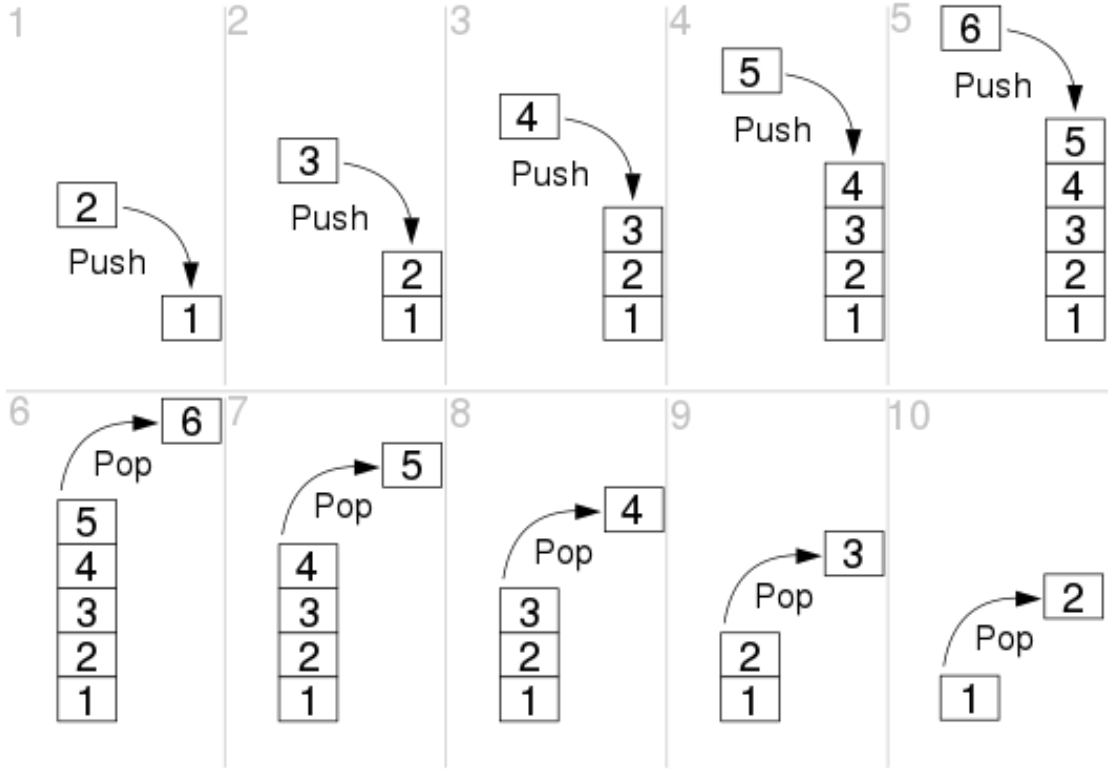


# OPERASI-OPERASI PADA STACK



- Operasi-operasi utama (**mandatory**) pada stack adalah :
  - **Push** : Proses menambahkan elemen baru di posisi Top.
  - **Pop** : Proses mengambil elemen yang berada di posisi Top.






Sumber : https://upload.wikimedia.org/wikipedia/commons/thumb/e/e4/Lifo_stack.svg/525px-Lifo_stack.svg.png

# OPERASI-OPERASI PADA STACK



- Operasi-operasi tambahan (opsional) pada stack adalah :
  - Initialize : Proses menginisialisasi stack sebelum digunakan.
  - isEmpty : Proses melihat status stack apakah kosong atau sudah mempunyai elemen
  - isFull : Proses melihat status stack apakah sudah penuh (Top = Max). Ini hanya digunakan pada stack yang diimplementasikan dengan array statis.
  - Count/Size : Proses menghitung berapa elemen yang ada dalam sebuah stack.
  - Peek : Proses melihat elemen data yang berada pada posisi Top tanpa melakukan Pop.
  - Display : Proses menampilkan isi stack tanpa melakukan pop.
  - Empty : Proses mengosongkan stack.

01158 - ALGORITMA DAN STRUKTUR DATA 2

# STACK DENGAN PYTHON

OLEH : ANDRI HERYANDI, M.T.

# IMPLEMENTASI STACK DENGAN PYTHON



- Jika anda ingin mengimplementasikan stack dengan menggunakan python, maka ada 2 cara yang bisa anda lakukan, yaitu :
  - Stack menggunakan class/struktur data bawaan dari python
  - Stack menggunakan class/struktur data buatan sendiri (*User-Defined-Class / User-Defined-Data-Structure*).

# IMPLEMENTASI STACK MENGGUNAKAN CLASS/STRUKTUR DATA BAWAAN PYTHON

OLEH : ANDRI HERYANDI, M.T.

# STACK DENGAN PYTHON



- Ada banyak cara mengimplementasikan stack menggunakan python.
- 2 diantaranya adalah :
  - Menggunakan list
  - Menggunakan class Collections.deque

# STACK DENGAN PYTHON (MENGGUNAKAN LIST)



- List mempunyai kemampuan untuk bekerja menjadi stack.
- Berikut adalah perbandingan methodnya :

| Operasi Stack | Method List |
|---|---|
| Push | append() |
| Pop | pop()<br>(tanpa parameter) |

# STACK DENGAN PYTHON (MENGGUNAKAN LIST)



- Contoh stack dengan menggunakan list.

```
stack = [] # stack kosong
print("Isi #1: ",stack," <-- TOP")
stack.append(1) # push(1)
stack.append(2) # push(2)
stack.append(3) # push (3)
print("Isi #2: ",stack," <-- TOP")
data = stack.pop() # pop #1
print("Hasil POP #1 : ", data) # view hasil pop #1
print("Hasil POP #2 : ", stack.pop()) # pop #2, view data
print("Isi #3: ",stack," <-- TOP")
```

## Hasil Run :

```
Isi #1:  []  <-- TOP
Isi #2:  [1, 2, 3]  <-- TOP
Hasil POP #1 :  3
Hasil POP #2 :  2
Isi #3:  [1]  <-- TOP
```

# STACK DENGAN PYTHON (MENGGUNAKAN COLLECTIONS.DEQUE)



- Class **deque** (*Double Ended Queue*) di module **collections** juga mempunyai kemampuan untuk bekerja sebagai stack.
- Berikut adalah perbandingan methodnya :

| Operasi Stack | Method List |
|---|---|
| Push | append() |
| Pop | pop()<br>(tanpa parameter) |

# STACK DENGAN PYTHON (MENGGUNAKAN LIST)



- Contoh stack dengan menggunakan class **collections.deque**.

```
from collections import deque
stack = deque() # stack kosong
print("Isi #1: ",stack," <-- TOP")
stack.append(1) # push(1)
stack.append(2) # push(2)
stack.append(3) # push (3)
print("Isi #2: ",stack," <-- TOP")
data = stack.pop() # pop #1
print("Hasil POP #1 : ", data) # view hasil pop #1
print("Hasil POP #2 : ", stack.pop()) # pop #2,view
data
print("Isi #3: ",stack," <-- TOP")
```

## Hasil Run :

```
Isi #1:  deque([])  <-- TOP
Isi #2:  deque([1, 2, 3])  <-- TOP
Hasil POP #1 :  3
Hasil POP #2 :  2
Isi #3:  deque([1])  <-- TOP
```

01158 - ALGORITMA DAN STRUKTUR DATA 2

# IMPLEMENTASI STACK MENGGUNAKAN CLASS/STRUKTUR DATA BUATAN SENDIRI

OLEH : ANDRI HERYANDI, M.T.

# STRUKTUR DATA PADA STACK

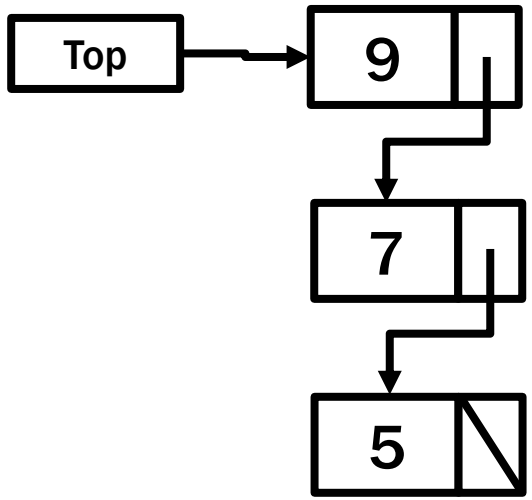


- Struktur Data Stack diimplementasikan menggunakan Array
- Struktur Data Stack diimplementasikan menggunakan Linked List.

Top=2

| Index | Isi |
|---|---|
| MAX-1 | |
| ... | |
| 3 | |
| 2 | 9 |
| 1 | 7 |
| 0 | 5 |

Stack menggunakan Array




Stack menggunakan Linked List

01158 - ALGORITMA DAN STRUKTUR DATA 2

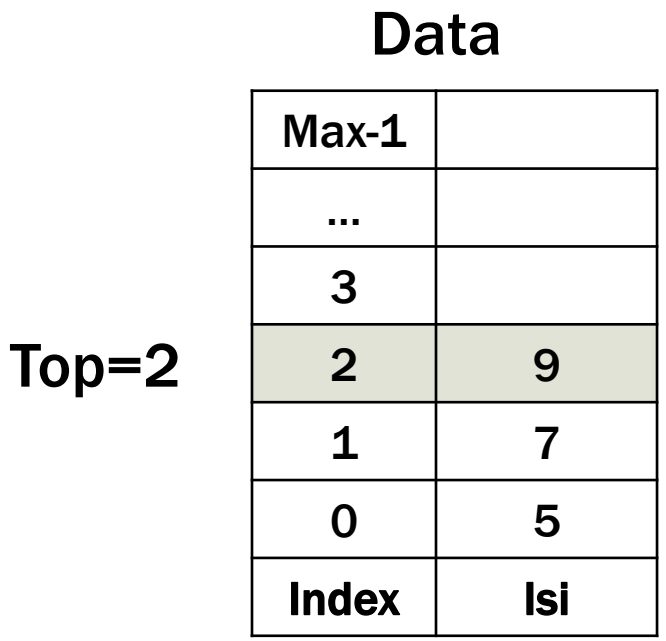
# STACK MENGGUNAKAN ARRAY

OLEH : ANDRI HERYANDI, M.T.

# STRUKTUR DATA STACK BUATAN SENDIRI (MENGGUNAKAN ARRAY)



- Struktur Data Stack yang diimplementasikan menggunakan Array terdiri dari 3 bagian yaitu :
  - Data : berupa array yang akan menampung isi stack
  - Top : berupa angka yang mencatat posisi sekarang
  - Max : berupa angka yang mencatat maksimal data yang dapat ditampung oleh stack.
- Struktur data array dalam python bisa menggunakan module array atau numpy.




Stack dengan Array

# STRUKTUR DATA STACK BUATAN SENDIRI (MENGGUNAKAN ARRAY)



- Pendeklarasian class struktur data untuk stack menggunakan array (Bahasa Python).

```
import array as ar

class stackArray:
    def __init__(self, max, datatype='i'):
        self.max = max # batas maksimal data dalam stack
        self.top = -1 # initialisasi top
        self.data = ar.array(datatype,[0]*max) # inisialisasi isi stack
```

- Pendeklarasian variable untuk stack menggunakan array (Bahasa Python).

```
stack = stackArray(10) # Stack of integer
stack2 = stackArray(10,'d') # Stack of double
```

# INISIALISASI STACK (ARRAY)



- Inisialisasi stack adalah proses mempersiapkan sebuah stack untuk digunakan.
- Proses yang dilakukan adalah dengan men-set nilai -1 pada variable/field yang menunjuk kepada nilai Top. Hal ini dilakukan agar penambahan data berikutnya dilakukan pada index array ke-0 (karena array di python dimulai dari index 0).
- **Catatan** : Jika anda menggunakan bahasa pemrograman yang menggunakan index awal bukan 0 maka nilai ini bisa disesuaikan (misalnya seting nilai top dengan 0 kalau index awal array dimulai dari 1).
- Dalam contoh ini, operasi inisialisasi dilakukan pada constructor class (agar setiap ada pembuatan objek, maka inisialisasi ini akan otomatis dipanggil).

# PEMERIKSAAN STACK KOSONG (ARRAY)



- Pemeriksaan ini digunakan untuk memeriksa apakah stack dalam keadaan kosong.
- Sebuah stack disebut kosong jika Top dari stack mempunyai nilai -1 (sesuai di proses inisialisasi stack).
- Pemeriksaan ini akan bisa digunakan dalam proses pop, karena proses pop hanya bisa dilakukan jika stack tidak kosong.
- **Catatan** : Jika index array anda dimulai dari 1 maka sebuah stack disebut kosong jika Top dari stack mempunyai nilai 0 (sesuai di proses inisialisasi stack).

# PEMERIKSAAN STACK KOSONG (ARRAY)



- Method **isEmpty**

```
def isEmpty(self):
        return self.top == -1;
```

# PEMERIKSAAN STACK PENUH (ARRAY)



- Pemeriksaan ini digunakan untuk memeriksa apakah stack dalam keadaan penuh.
- Sebuah stack disebut penuh jika Top dari stack mempunyai nilai sama dengan maksimum element array dikurangi 1.
- Pemeriksaan ini akan bisa digunakan dalam proses push, karena proses push hanya bisa dilakukan jika stack tidak penuh.
- Pemeriksaan stack penuh hanya dilakukan jika stack diimplementasikan menggunakan array.
- **Catatan** : Pada array yang indexnya dimulai dari 1, maka stack penuh adalah Ketika nilai top sama dengan nilai maksimum.

# PEMERIKSAAN STACK PENUH (ARRAY)



- Method **isFull**

```
def isFull(self):
        return self.top == self.max -1
```

# BANYAK ELEMEN DALAM STACK (ARRAY)



- Proses ini digunakan untuk mengetahui berapa banyak elemen yang berada dalam stack.
- Pada stack yang diimplementasikan dengan array, banyaknya elemen stack dapat dilihat dari nilai Top dari stack + 1. Contoh : Ketika nilai top = 0, maka banyak data adalah 1 dst.
- **Catatan** : Jika array anda dimulai dari 1, maka banyak elemen dari stack adalah nilai top dari stack.

# BANYAK ELEMEN DALAM STACK (ARRAY)



- Method **count**

```
def count(self):
        return self.top + 1
```

# MENAMPILKAN INFORMASI STACK (ARRAY)



- Proses ini digunakan untuk menampilkan informasi stack.
- Informasi yang ditampilkan adalah data, top (kalau dibutuhkan) dan nilai maksimalnya (kalau dibutuhkan)
- Dalam python, gunakan function `__str__` dalam class agar dieksekusi ketika membutuhkan output suatu class dalam bentuk string.

# MENAMPILKAN ELEMEN STACK (ARRAY)



- Method untuk menampilkan informasi stack **__str__**

```
def __str__(self):
        out = "stackArray(Data : ["
        if self.isEmpty():
            out = out + " EmptyStack "
        else:
            for i in range(self.top + 1):
                out = out + str(self.data[i])+ " "
        out = out + "]"
        out = out + ", Top : " + str(self.top)
        out = out + ", Max : " + str(self.max)
        out = out + ")"
        return out
```

# MENAMBAH ELEMEN STACK (ARRAY)

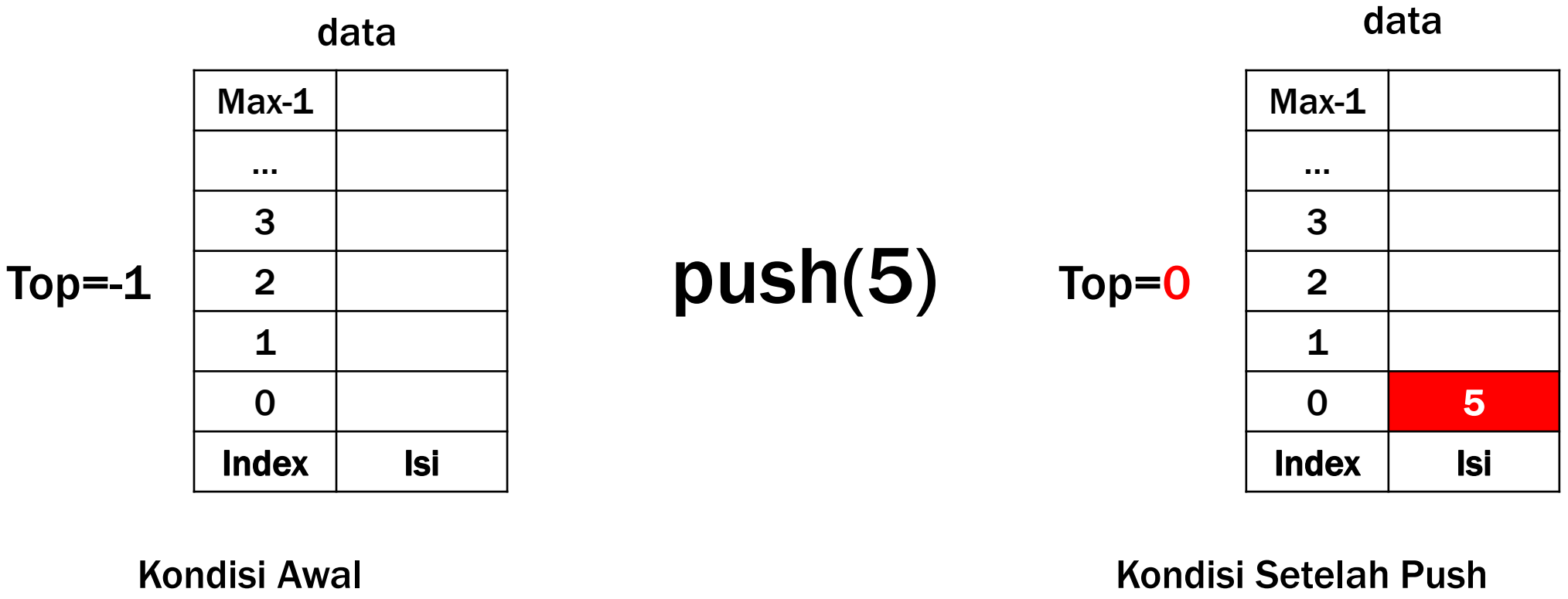


- Proses ini sering disebut **Push**.
- Operasi Push digunakan untuk menambahkan elemen baru di posisi Top.
- Langkah-langkah untuk mengambah elemen stack adalah :
  1. Lakukan pemeriksaan apakah stack sudah penuh. Jika stack sudah penuh maka tampilkan pesan bahwa “Stack penuh” (*overflow error report*). Jika stack belum penuh maka lakukan Langkah 2 dan Langkah 3.
  2. Nilai Top dari stack ditambah 1
  3. Isikan data baru pada element stack pada posisi Top yang baru.

# MENAMBAH ELEMEN STACK (ARRAY)



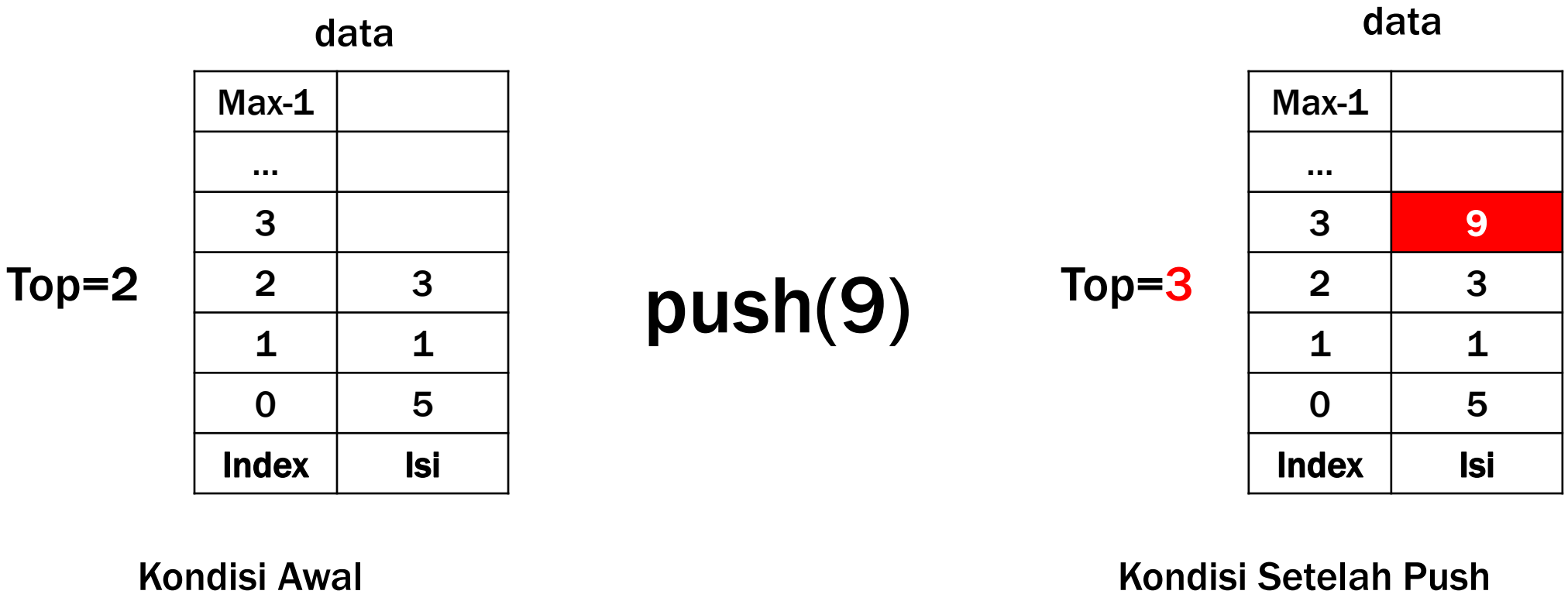
- Ilustrasi push ketika stack masih kosong



# MENAMBAH ELEMEN STACK (ARRAY)



- Ilustrasi push ketika stack sudah memiliki data



# MENAMBAH ELEMEN STACK (ARRAY)



- Method **push**

```
def push(self, data):
        if self.isFull():
            print("Stack Penuh (Overflow)")
        else:
            self.top = self.top + 1
            self.data[self.top] = data
```

# MENGAMBIL/MENGHAPUS ELEMEN STACK (ARRAY)

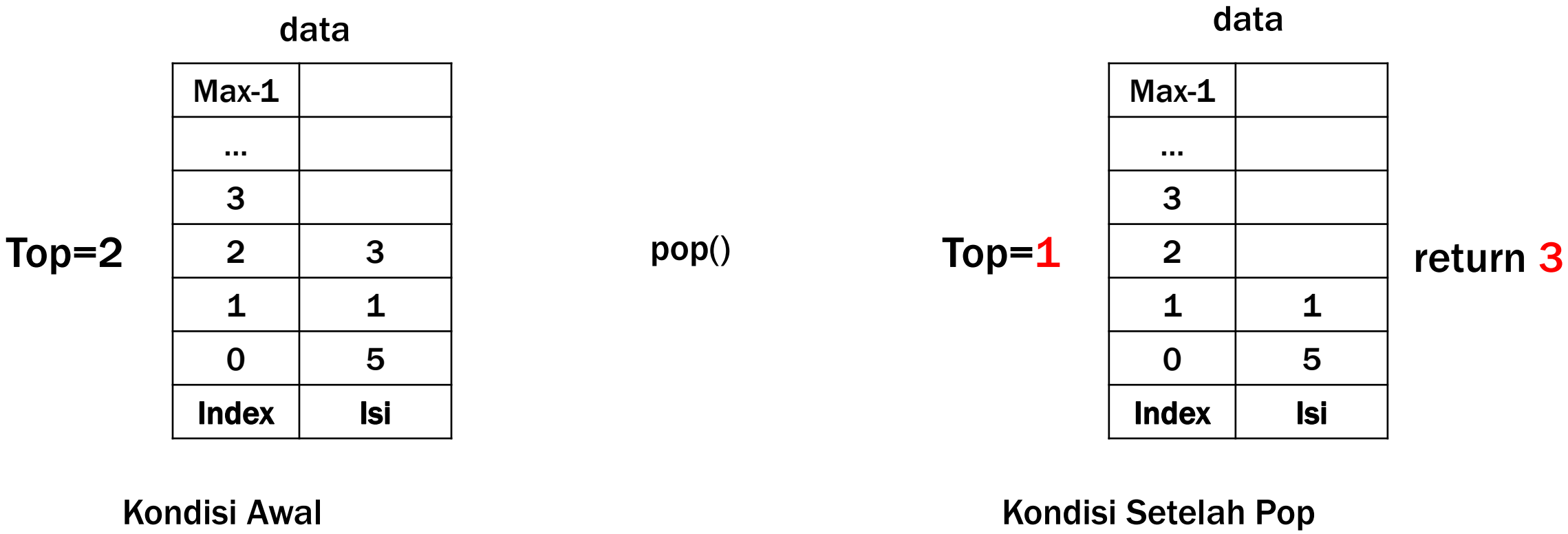


- Proses ini sering disebut **Pop**.
- Operasi Pop digunakan untuk mengambil elemen yang berada di posisi Top.
- Langkah-langkah untuk mengambah elemen stack adalah :
  1. Lakukan pemeriksaan apakah stack kosong. Jika stack kosong maka tampilkan pesan bahwa "Stack kosong" (*Underflow Error Report*). Jika stack tidak kosong maka lakukan Langkah 2 dan Langkah 3.
  2. Ambil nilai yang ditunjuk oleh Top (asumsi nama variable adalah `dataInTop`)
  3. Nilai Top stack dikurangi 1
  4. Returnkan nilai `dataInTop`.

# MENGAMBIL/MENGHAPUS ELEMEN STACK (ARRAY)



- Ilustrasi pop ketika stack sudah memiliki data



# MENGAMBIL/MENGHAPUS ELEMEN STACK (ARRAY)



- Method **pop**

```
def pop(self):
        if self.isEmpty():
            print("Stack Kosong (Underflow)")
            return None
        else:
            dataInTop = self.data[self.top]
            self.top = self.top - 1
            return dataInTop
```

# MELIHAT NILAI PADA POSISI TERAKHIR (ARRAY)



- Proses ini sering disebut **Peek**.
- Operasi Peek digunakan untuk melihat element yang berada di posisi Top tanpa melakukan Pop (hanya lihat saja, tanpa mengambilnya).

# MELIHAT NILAI PADA POSISI TERAKHIR (ARRAY)



- Method **peek**

```
def peek(self):
        if self.isEmpty():
            print("Stack Kosong (Underflow)")
            return None
        else:
            return self.data[self.top]
```

# MENGOSONGKAN STACK (ARRAY)



- Operasi ini digunakan untuk mengatur stack agar kondisinya dianggap kosong.
- Ada 2 cara untuk mengosongkan stack, yaitu :
  1. Langsung mengisi top dari stack menjadi -1. Cara ini hanya bisa digunakan di stack array. Cara ini adalah paling cepat.
  2. Lakukan pengulangan pop sampai stacknya kosong. Langkah ini bisa digunakan di stack yang diimplementasikan dengan array atau pun dengan linked list.
- Cara ke-1 akan digunakan karena kita sedang menggunakan stack yang menggunakan Array.

# MENGOSONGKAN STACK (ARRAY)



- Method **empty**

```
def empty(self):
        self.top = -1
```

# CONTOH PENGGUNAAN STACK (ARRAY)



- Contoh penggunaan (#part 1):

```
stack = stackArray(5,"i")
print("#01 : ", stack)
stack.push(5)
stack.push(10)
stack.push(15)
stack.push(20)
stack.push(30)
print("#02 : ", stack)
stack.push(50)
print("#03 : ", stack)

<lihat halaman selanjutnya>
```

```
#01 :  stackArray(Data : [ EmptyStack ], Top : -1, Max : 5)
#02 :  stackArray(Data : [5 10 15 20 30 ], Top : 4, Max : 5)
Stack Penuh (Overflow)
#03 :  stackArray(Data : [5 10 15 20 30 ], Top : 4, Max : 5)
```

# CONTOH PENGGUNAAN STACK (ARRAY)



- Contoh penggunaan (#part 2):

```
data = stack.pop() # POP #1
print("Setelah POP #1 ")
print("#04 : Data :", data)
print("#05 :",stack)
print("#06 : Peek:",stack.peek())
print("#07 : POP #2:", stack.pop())
print("#08 : POP #3:", stack.pop())
print("#09 :", stack)
stack.empty()
print("#10 :", stack)
print("#11 : POP #4:", stack.pop())
```

```
Setelah POP #1
#04 : Data : 30
#05 : stackArray(Data : [5 10 15 20 ], Top : 3, Max : 5)
#06 : Peek: 20
#07 : POP #2: 20
#08 : POP #3: 15
#09 : stackArray(Data : [5 10 ], Top : 1, Max : 5)
#10 : stackArray(Data : [ EmptyStack ], Top : -1, Max : 5)
Stack Kosong (Underflow)
#11 : POP #4: None
```

01158 - ALGORITMA DAN STRUKTUR DATA 2

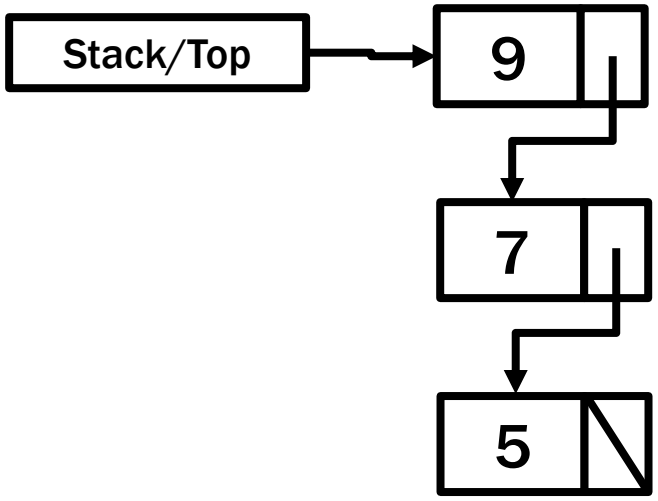
# STACK MENGGUNAKAN LINKED LIST

OLEH : ANDRI HERYANDI, M.T.

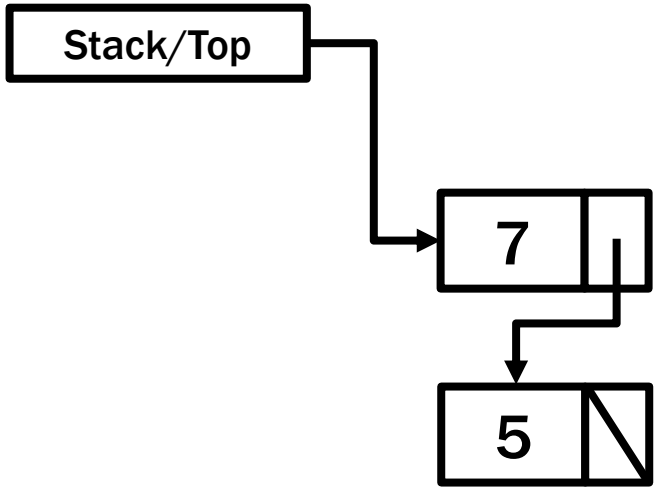
# PRESENTASI STACK DENGAN LINKED LIST DI MEMORI



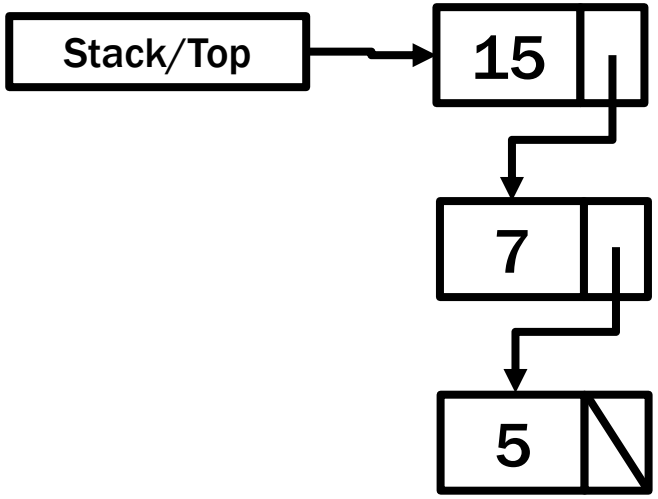
- Stack dapati diimplementasikan dengan menggunakan linked list dengan cara setiap penambahan dan penghapusan data hanya dilakukan di node depan.
- Node depan boleh diberi nama Stack atau Top.




Kondisi Awal




Setelah Penghapusan




Setelah Penambahan

# STRUKTUR DATA (LINKED LIST)



- Pendeklarasian struktur data untuk stack menggunakan linked list (Bahasa Python).

```
class Node:
    def __init__(self, data):
        self.info = data
        self.next = None

class stackLinkedList:
    def __init__(self):
        self.top = None
```

Mirip dengan pendeklarasian linked list, hanya mengubah pointer awal menjadi top.

- Pendeklarasian variable untuk stack menggunakan linked list (Bahasa Python).

```
stack = stackLinkedList()
```

# INISIALISASI STACK (LINKED LIST)



- Inisialisasi stack adalah proses mempersiapkan sebuah stack untuk digunakan.
- Proses yang dilakukan adalah memberikan nilai NIL (Python = None) kepada top dari stack yang menandakan bahwa stack kosong.
- Karena stack ini dibuat dalam bentuk Class, maka proses inisialisasi dimasukkan ke dalam construktur class agar setiap pembuatan suatu stack akan otomatis memanggil inisialisasi ini.

# PEMERIKSAAN STACK KOSONG (LINKED LIST)



- Pemeriksaan ini digunakan untuk memeriksa apakah stack dalam keadaan kosong.
- Sebuah stack disebut kosong jika stack/top menunjukan nilai NIL.
- Pemeriksaan ini akan bisa digunakan dalam proses pop, karena proses pop hanya bisa dilakukan jika stack tidak kosong.

# PEMERIKSAAN STACK KOSONG (LINKED LIST)



- Method **isEmpty**

```
def isEmpty(self):
        return self.top is None
```

# PEMERIKSAAN STACK PENUH (LINKED LIST)



**Tidak ada pemeriksaan stack penuh jika stack menggunakan linked list, karena linked list tidak mempunyai batasan nilai maximal**

# BANYAK ELEMEN DALAM STACK (LINKED LIST)



- Proses ini digunakan untuk mengetahui berapa banyak elemen yang berada dalam stack.
- Pada stack yang diimplementasikan dengan linked list, banyak data bisa didapatkan dengan menelusuri linked list dari node pertama (top) sampai node terakhir (node yang next-nya bernilai Nil).

# BANYAK ELEMEN DALAM STACK (LINKED LIST)



- Method **count**

```
def count(self):
        cnt = 0
        curNode = self.top
        while curNode is not None:
            cnt = cnt + 1
            curNode = curNode.next
        return cnt
```

# MENAMPILKAN ELEMEN STACK (LINKED LIST)



- Proses ini digunakan untuk menampilkan informasi stack (isi dan posisi top)
- Menampilkan isi stack dalam linked list dilakukan dengan melakukan penelusuran linked list untuk menampilkan elemen linked list satu per satu.
- Dalam python, gunakan function `__str__` dalam class agar dieksekusi ketika membutuhkan output suatu class dalam bentuk string.

# MENAMPILKAN ELEMEN STACK (LINKED LIST)



- Method `__str__`

```
def __str__(self):
        out = "stackLinkedList(Data : ["
        if self.isEmpty():
            out = out + " EmptyStack "
        else:
            out = out + " TOP -> "
            curNode = self.top
            while curNode is not None:
                out = out + str(curNode.info) + " "
                curNode = curNode.next
        out = out + "]"
        out = out + ", Count : " + str(self.count())
        out = out + ")"
        return out
```

# MENAMBAH ELEMEN STACK (LINKED LIST)

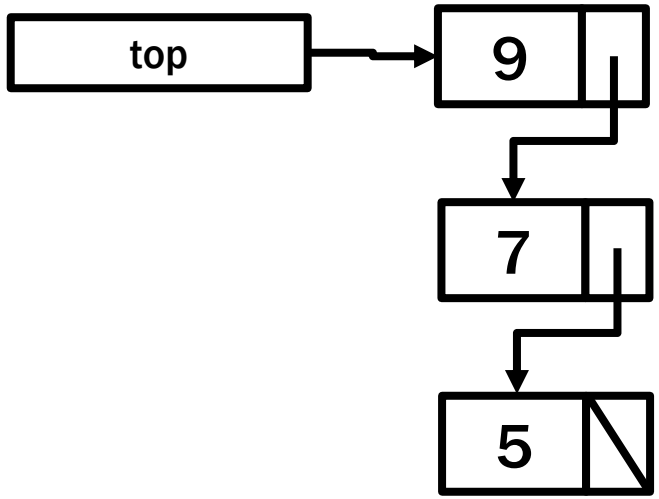


- Proses ini sering disebut **Push**.
- Operasi Push digunakan untuk menambahkan elemen baru di posisi Top.
- Dalam stack yang menggunakan linked list, proses push dilakukan dengan cara melakukan penambahan node di awal linked list.

# MENAMBAH ELEMEN STACK (LINKED LIST)

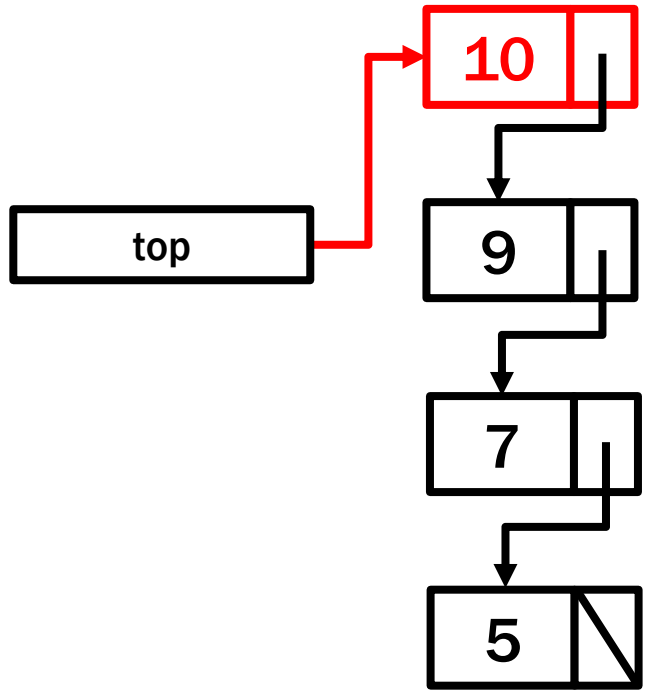


- Ilustrasi Push dalam stack




Kondisi Awal

Push(10)




Setelah Penambahan

# MENAMBAH ELEMEN STACK (LINKED LIST)



- Method **push**

```
def push(self, data):
        newNode = Node(data)
        newNode.next =  self.top
        self.top = newNode
```

# MENGAMBIL/MENGHAPUS ELEMEN STACK (LINKED LIST)

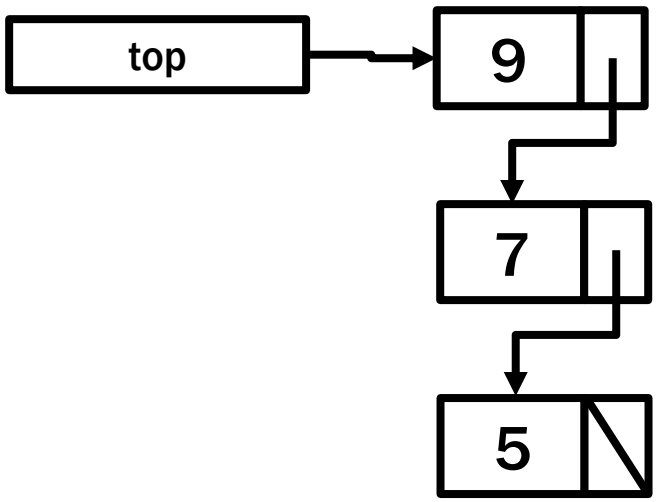


- Proses ini sering disebut **Pop**.
- Operasi Pop digunakan untuk mengambil elemen yang berada di posisi Top.
- Pada stack yang menggunakan linked list, proses pop sama saja dengan proses penghapusan node di awal linked list.
- **Catatan** : Lakukan pemeriksaan apakah stack kosong sebelum melakukan pop. Pop hanya bisa kalau stack tidak kosong.

# MENGAMBIL/MENGHAPUS ELEMEN STACK (LINKED LIST)

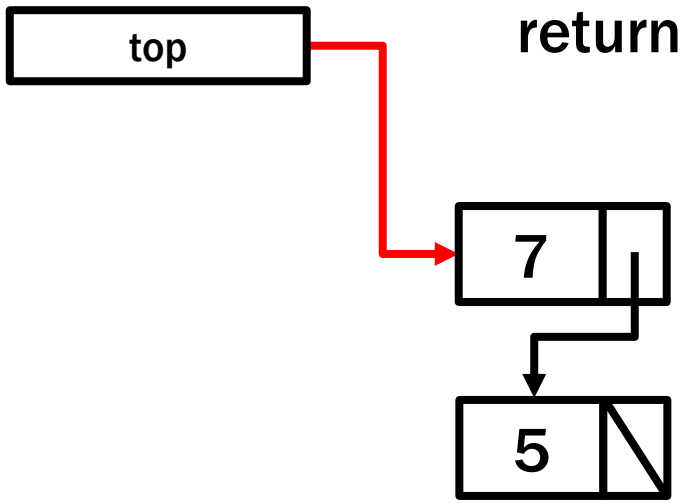


- Ilustrasi Pop dalam stack




Kondisi Awal

pop()




Setelah Pengambilan

# MENGAMBIL/MENGHAPUS ELEMEN STACK (LINKED LIST)



- Method **pop**

```
def pop(self):
        if self.isEmpty():
            print(" POP Error. Empty Stack")
            return None
        else:
            prevTop = self.top
            dataInTop = self.top.info
            self.top = self.top.next
            del prevTop
            return dataInTop
```

# MELIHAT NILAI PADA POSISI TERAKHIR (LINKED LIST)



- Proses ini sering disebut **Peek**.
- Operasi Peek digunakan untuk melihat element yang berada di posisi Top tanpa melakukan Pop (hanya lihat saja, tanpa mengambilnya).

# MELIHAT NILAI PADA POSISI TERAKHIR (LINKED LIST)



- Method **peek**

```
def peek(self):
        if self.isEmpty():
            print(" POP Error. Empty Stack")
            return None
        else:
            return self.top.info
```

# MENGOSONGKAN STACK (LINKED LIST)



- Operasi ini digunakan untuk mengatur stack agar kondisinya dianggap kosong.
- Ada 2 cara untuk mengosongkan stack yaitu :
  1. Langsung mengisi top dari stack menjadi 0. Cara ini hanya bisa digunakan di stack array.
  2. Lakukan pengulangan pop sampai stacknya kosong. Langkah ini bisa digunakan di stack yang diimplementasikan dengan array atau pun dengan linked list.
- Cara ke-2 akan digunakan karena kita sedang menggunakan stack yang menggunakan Linked List.

# MENGOSONGKAN STACK (LINKED LIST)



- Method **empty**

```
def empty(self):
        while not self.isEmpty():
            self.pop()
```

# CONTOH PENGGUNAAN STACK (LINKED LIST)



- Contoh penggunaan (#part 1):

```
stack = stackLinkedList()
print("#01 : ", stack)
stack.push(5)
stack.push(10)
stack.push(15)
stack.push(20)
stack.push(30)
print("#02 : ", stack)
stack.push(50)
print("#03 : ", stack)
```

```
#01 :  stackLinkedList(Data : [ EmptyStack ], Count : 0)
#02 :  stackLinkedList(Data : [ TOP -> 30 20 15 10 5 ], Count : 5)
#03 :  stackLinkedList(Data : [ TOP -> 50 30 20 15 10 5 ], Count : 6)
```

# CONTOH PENGGUNAAN STACK (LINKED LIST)



- Contoh penggunaan (#part 2):

```
data = stack.pop() # POP #1
print("Setelah POP #1 ")
print("#04 : Data :", data)
print("#05 :",stack)
print("#06 : Peek:",stack.peek())
print("#07 : POP #2:", stack.pop())
print("#08 : POP #3:", stack.pop())
print("#09 :", stack)
stack.empty()
print("#10 :", stack)
print("#11 : POP #4:", stack.pop())
```

```
Setelah POP #1
#04 : Data : 50
#05 : stackLinkedList(Data : [ TOP -> 30 20 15 10 5 ],
Count : 5)
#06 : Peek: 30
#07 : POP #2: 30
#08 : POP #3: 20
#09 : stackLinkedList(Data : [ TOP -> 15 10 5 ], Count :
3)
#10 : stackLinkedList(Data : [ EmptyStack ], Count : 0)
 POP Error. Empty Stack
#11 : POP #4: None
```

# 01158 - ALGORITMA DAN STRUKTUR DATA 2

# LATIHAN

OLEH : ANDRI HERYANDI, M.T.

# CONTOH PENGGUNAAN STACK (ARRAY)



- Konversikan bilangan desimal ke biner menggunakan stack.
- Algoritma :
  1. Input bilangan decimal
  2. Ulangi Langkah 3 s.d 4 sampai nilai bilangan desimal menjadi 0.
  3. Bagi bilangan desimal dengan angka 2, sisanya push ke stack
  4. Bilangan desimal dibagi 2 menggunakan div
  5. Lakukan pengulangan Langkah 6 dan 7 sampai stack kosong
  6. Pop nilai dari stack, simpan di sebuah variable
  7. Tampilkan variable tersebut

```
Desimal : 29   [input]
Biner   : 11101 [output]
```

# CONTOH PENGGUNAAN STACK (ARRAY)



- Proses konversi bilangan decimal ke biner (Contoh desimal 29).
  1. Stack([Kosong])
  2. Lakukan pembagian nilai desimal dengan angka 2 sampai hasil pembagiannya sama dengan 0. Prosesnya:
     a. 29 div 2 = 14, sisa 1. Push sisa ke stack sehingga isi Stack([TOP → 1])
     b. 14 div 2 = 7, sisa 0, Push sisa ke stack sehingga isi stack([TOP → 0 1])
     c. 7 div 2 = 3, sisa 1, Push sisa ke stack sehingga isi stack([TOP → 1 0 1])
     d. 3 div 2 = 1, sisa 1, Push sisa ke stack sehingga isi stack([TOP → 1 1 0 1]
     e. 1 div 2 = 0, sisa 1, Push sisa ke stack sehingga isi stack([TOP → 1 1 1 0 1])
     f. Karena hasil pembagian sudah 0 maka proses pembagian dengan 2 telah selesai.
  3. Hasil didapat dengan melakukan pop sampai stack kosong. Setiap hasil top tambahkan ke biner.
     a. Pop #1 menghasilkan 1, maka biner adalah 1
     b. Pop #2 menghasilkan 1, maka biner adalah 11
     c. Pop #3 menghasilkan 1, maka biner adalah 111
     d. Pop #4, menghasilkan 0, maka biner adalah 1110
     e. Pop #5, menghasilkan 1, maka biner adalah 11101
     f. Stack Kosong. Konversi selesai.

# SEKIAN

**Ada pertanyaan?**

**Silahkan tanyakan melalui Group Whatsapp, email, LMS.**

# FORUM DISKUSI




**LMS UNIKOM**
**https://lms.unikom.ac.id**

**Group Whatsapp Perkuliahan**

**Youtube Comments**
https://unikom.id/YTCStrukturData