Iterators

Func iterator
Abdul

Iterators

↳ Iter(iterable) gives an iterator on iterable

↳ next(iterator) gives an element and moves next

```
L1 = [5,6,10,11,15]
for i in L1:
    pt(i,end=" ")
# 5 6 10 11 15
```

```
L1=[5,6,10,11,15]
it=iter(L1)
pt(it)
<list_operator>
pt(next(it))
# 5
pt(next(it))
# 6
pt(next(it))
# 10, 11, 15
# StopIteration
```

↳ L1=[5,6,10,11,15]

```
for i in range(len(L1)+1):
    pt(next(it),end=" ")
```

5,6,10,11,15 // StopIteration

↳ Tuples

```
T1=[5,6,7]
it=iter(T1)
for i in range(len(T1)):
    pt(next(it),end=" ")
```

5,6,7

sets

```
s1 = {5, 6, 7}
it = iter(s1)
for i in range(len(s1)):
    print(next(s1), end=" ")
```

TypeError

Dictionary

```
d1 = {1: "one", 2: "two", 3: "three"}
it = iter(d1)
for i in range(d1):
    print(next(it), end=" ")   # 1 2 3
```

Strings

```
s1 = "NNR"
it = iter(s1)
for i in range(len(s1)):
    print(next(it), end=" ")
```

N N R

range

```
r = range(3, 6)
it = iter(r)
for i in range(len(r)):
    print(next(it), end=" ")
```

3 4 5

next(iterator) gives an element / moves

Variable length x

pt()
pt(10)
pt(10,12.5)
pt(10,12.5,"hello", 5, True)

→ print function can take Variable length arguments

ABDUL
fun 2

Variable length - 1
Positional

def fun(*args):
1. *args → is used for Variable length arguments
2. Tuple is created for Variable length arguments

def fun(*args):
    pt(args)
    # pt(type(args)) #Tuple
    for i in args:
        pt(i, end=" ")

fun(5) #(5)
fun(5,10) # (5,10)
fun(5,10,15) # (5,10,15)
fun(5,1.25,"hello",15) # 5, 1.25, "hello" 15

Tuple

Prog 2

def fun(*args):
    for x in args:
        if (type(x) == int):
            pt(x)

fun(10,12.5,"hello", True, 3+4j, 15)
# 10,15

fun(a,b,*args)
a and b are positional args only

```
def fun(a,b,*args):
    pt(a,b,args)

fun(10,20,30)
    # 10,20,(30,)
fun(10,20,30,40,50)
    10,20,(30,40,50)
fun(a=10,b=20,30,40,50)  X
    Syntax Error
```

↳ fun(*args,a,b)
a,b are keyword args only

prob 1:-

```
def fun(*args,a,b):
    pt(a,b,args)
fun(10,20,30,40,50)
    #TypeError
fun(10,20,30,a=40,b=50)
    # 40,50,(10,20,30)
```

fun(*LI) → unpacking the list elements

```
def fun(*args):
    pt(args,len(args))
LI=[10,20,30]
fun(LI)  # ([10,20,30],1)
```

```
LI=[10,20,30]
fun(*LI)
#(10,20,30) 3
```

Generators

Assoul generators

(3)

```
r = range(4)
it = iter(it)
pt(it)
```

↳ range is example of generators
↳ Iterator is built in
↳ range is built-in

Prog 1:-

```
def myrange(n):
    i=0
    while(i<n):
        yield i
        i=i+1

m=myrange(n)
for i in range(n):
    pt(next(m), end=" ")
#0,1,2,3
pt(next(m))
#stopIteration
```

Prog 2

```
def mylist(d):
    i=0
    while True:
        yield d[i]
        i=(i+1)%7

d=['sun' ... ]
m=mylist(d)
for i in range(len
    pt(next(m) end
```

SUN Mon Tue ... THU

Local and Global

Local Variables

↳ Declared inside a function

Global Variables

↳ Declared outside of a function
↳ Can be read inside of functions
↳ Cannot be modified unless global is declared
↳ Must be declared before function call

↳ locals() → Gives a dictionary of local variables

↳ globals() → Gives a dictionary of global variables.

a.
```
def fun():
    a=10
    pt(a)
fun()
```
10.

```
g=5.25
pt("outside:",g)
def fun():
    a=10
    pt(g)
fun()
```
outside: 5.25
10

c.
```
g=5.25
pt("outside:",g)
def fun():
    a=10
    pt("local:",a)
    pt("global:",g)
fun()
```
outside: 5.25
local: 10
global: 5.25

```
g=5.25
pt('outside:1',g)
def fun():
    a=10
    pt("local:",a)
    pt("global:",g)
fun()
pt('outside:2',g)
```

```
10
5.25
5.25
```

Prog 5:

```
g=5.25
pt('outside:1',g)
def fun():
    a=10
    g=199
    pt(a)
    pt(g)
fun()
pt("outside2:",g)
```

```
5.25
10
199
5.25
```

Prog 6

```
g=5.25
pt("1",g)
def fun():
    a=10
    global g
    g=199
    pt("local:",a)
    pt("global:",g)
fun()
pt('outside:2',g)
```

```
outside1:  5.25
local:  10
global:  199
outside2:  199
```

Prog 7

```
# g = 5.25
def func():
    a = 10
    print('local', g)
    print(global g)
g = 5.25
func()
g = 5.25
```

NameError:
name 'g' not defined

locals()

globals()

```
x, y, z = 5, 1.25, 'hi'
def fun():
    a, b, c = 1, 2, 3
    print(locals())
# {'a': 1, 'b': 2, 'c': 3}
    print(globals())
        'x': 5, 'y': 1.25, 'z': 'hi',
```

↳ locals() gives a dictionary of local variables

↳ globals() gives a dictionary of global variables

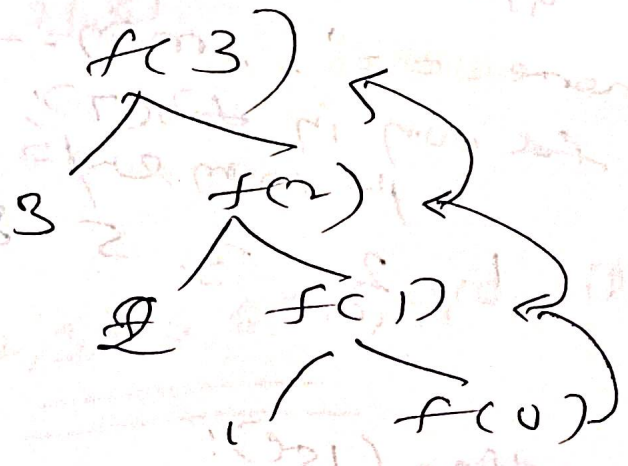
Ibrahim Abdul

Recursive :-

```
def fun(n):
    if(n>0):
        pt(n,end=" ")
        fun(n-1)

fun(3)
```

3 2 1



factorial

```
def fact(n):
    if(n<=0):
        return 1
    else:
        return n*fact(n-1)

pt(fact(5)) = 120.
```

f(5)
5 * fact(4) = 24
4 * fact(3) = 6
3 * fact(2) = 2
2 * fact(1) = 1
1 * fact(0) = 1

Fib

```
def fib(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else: return fib(n-1)+fib(n-2)

for i in range(8):
    pt(fib(i),end=" ")
```

0 1 1 2 3 5 8 13

```
def fib(n):
    a,b = 0,1
    for i in range(n+1):
        yield a
        a,b = b,a+b
if __name__ == '__main__':
    for num in fib(7):
        print(num,end=" ")
```

0 1 1 2 3 5 8 13

flatten

```
def flatten(lst):
    for item in lst:
        if hasattr(item,'__iter__'):
            # print(f'iterable {item}')
            yield from flatten(item)
        else:
            # print(f'{item} is not iterable')
            yield item
L=[1,2,[3,4,[5,6,7],8],9,[10,11]]
flat = flatten(L)
flat_list = list(flat)
print(flat_list)
```

[1, 2 [3, 4, [5, 6, 7], 8], 9, [10, 11]]

1 is not iterable
2 ''
iterable [3, 4, [5, 6, 7], 8]
3
4
iterable [5, 6, 7]
5
6
7
8
9
iterable [10, 11]
10 is not iterable
11 is not iterable

[1, 2, 3, 4, 5, 6, 7, 8, 9, [10, 11]]

Month calendar

```
import calendar
def next_month():
    count = 1
    while True:
        name = calendar.month_name[count]
        yield name
        count = count % 12 + 1
print(count)
it = next_month()
print(next(it))   # January
print(next(it))   # Feb
```

2 3 4 5 6