Functions & Exception Handling

🎯 Session Objectives:

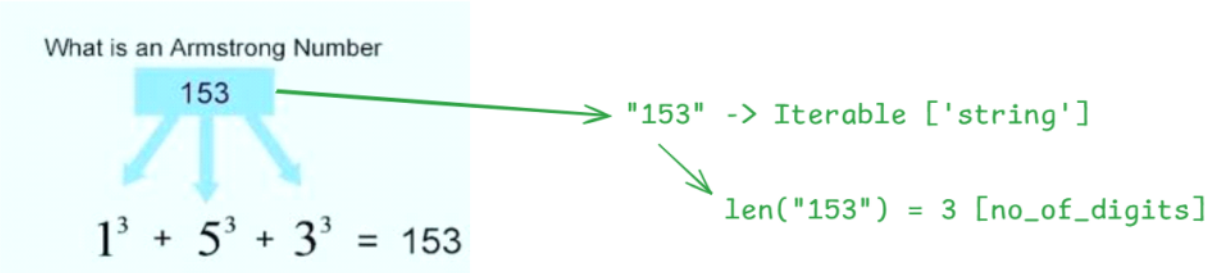
- ✅ Understand what functions are and why we use them.
- ✅ Learn to define functions, with parameters and arguments.
- ✅ Explore the use of the return statement.
- ✅ Understand scope and namespaces.
- ✅ Understand recursion
- ✅ Use lambda (anonymous) functions
- ✅ Understand Python's exception handling model
- ✅ Apply try, except, else, finally, and raise statements effectively

Python Searches in LEGB Order:

Level	Meaning
L	Local: inside current function
E	Enclosing: functions inside functions
G	Global: top-level script
B	Built-in: Python's built-in functions

Armstrong Number

sum of its own digits each raised to the power of the number of digits



$153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

$371 = 3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$

$9474 = 9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 + 2401 + 256 = 9474$

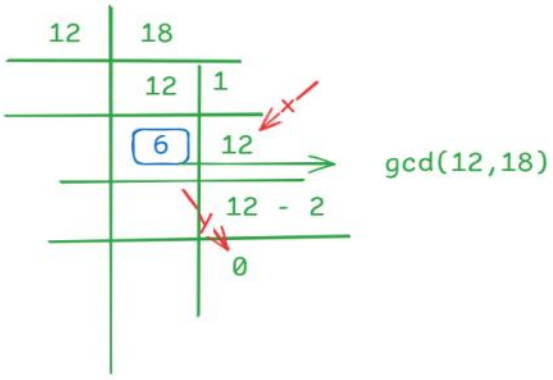
```python
def is_armstrong(val):
    str_val = str(val) # '153'
    num_digits = len(str_val) # 3
    sum_of_power = 0
    for digit in str_val: # ['1','5','3']
        sum_of_power += int(digit) ** num_digits
    return sum_of_power == val # Boolean Return

is_armstrong(153)
```

Memory

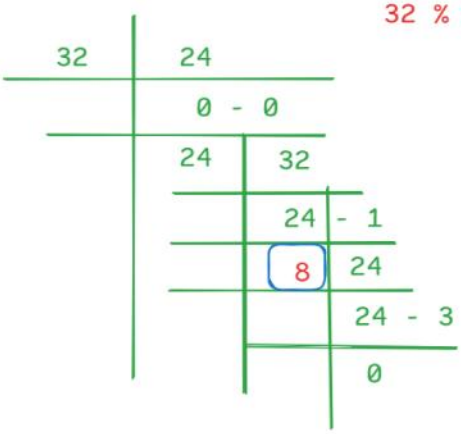
val = 153
str_val = '153'
num_digits = 3
sum_of_power = ~~0~~ ~~1~~ 126 153
digit = ~~'1'~~ ~~'5'~~ ~~'3'~~ 'None'

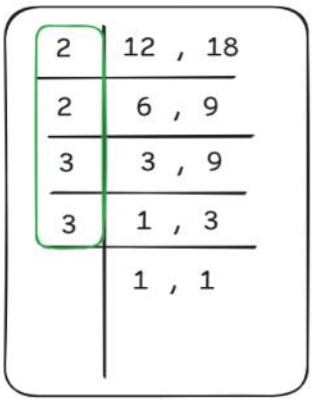
True

```
sum_of_power = sum_of_power + int(digit) ** num_digits
             = 0 + (1^3) = 1
             = 1 + (5^3) = 1 + 125 = 126
             = 126 + (3^3) = 126 + 27 = 153
```



32 % 24 = 8





2 * 2 * 3 * 3 = 4 * 9 = 36

LCM

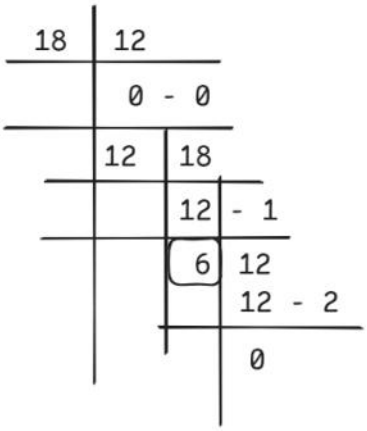
```python
# GCD [Greatest Common
def gcd(x,y):
    while y!=0 :
        x,y = y , x%y
    return abs(x)
gcd(12,18)
```

```
6
```

Memory

gcd(12,18)

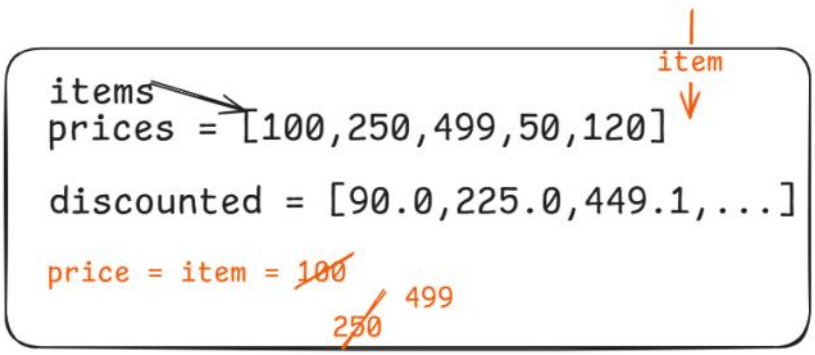
x = ~~12~~ ~~18~~ ~~12~~ 6
y = ~~18~~ ~~12~~ ~~6~~ 0



$$LCM(x,y) = abs(x*y) // gcd(x,y)$$

```python
def transform_list(func , items): # func , items -> iterables
    return [func(item) for item in items] # list Comprehension f(x)

def apply_discount(price):
    return round(price * 0.90, 2) # 10% Discount -> 0.90

def add_tax(price):
    return round(price * 1.05, 2) # 5% Standard Tax

prices = [100,250,499,50,120]
discounted = transform_list(apply_discount , prices)
taxed = transform_list(add_tax , discounted)

print("Original Price: " , prices)
print("Discounted Price: " , discounted)
print("Discounted Price with taxes: " , taxed)
```

```
Original Price:  [100, 250, 499, 50, 120]
Discounted Price:  [90.0, 225.0, 449.1, 45.0, 108.0]
Discounted Price with taxes:  [94.5, 236.25, 471.56, 47.25, 113.4]
```



```python
# Global Scope Vs Local Scope
x = 10 # 'global'
def greet():
    y = 15 # 'local'
    print(x) # 10
    print(y) # 15
greet()
```

```
10
15
```

Memory

```
x = 10
y = 15
```

Console

```
10
15
```

```python
# Global Scope Vs Local Scope
x = 10 # 'global'
def greet():
    y = 15 # 'local'
    print(x) # 10
    print(y) # 15
greet()
print(x) # 10
# print(y) # NameError: name 'y' is not defined
```

```
10
15
10
```

Memory

```
x = 10
~~y = 15~~
```

```
10
15
10
NameError
```

```python
gesture = 'Happy' # 'global'
def how_you_feel():
    gesture = "worried" # 'local'
    print(f"I'm feeling {gesture}") # 'Worried'

print(f"I'm feeling {gesture}") # 'Happy'
how_you_feel() # 'Worried'
print(f"I'm feeling {gesture}") # 'Happy'
```

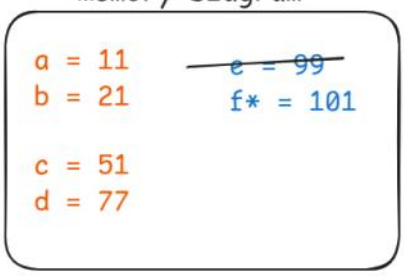
```
I'm feeling Happy
I'm feeling worried
I'm feeling Happy
```

Memory

```
gesture = 'Happy'
```

```python
a = 11
b = 21
def outer_fx():
    c = 51
    d = 77
    def inner_fx():
        e = 99
        global f
        f = 101
        print(e+f) # 200
    inner_fx()
    print(f) # 101
    print(b) # 21
    print(c) # 51
    # print(e) # NameError
outer_fx()
print(f) # 101
```

Memory Diagram

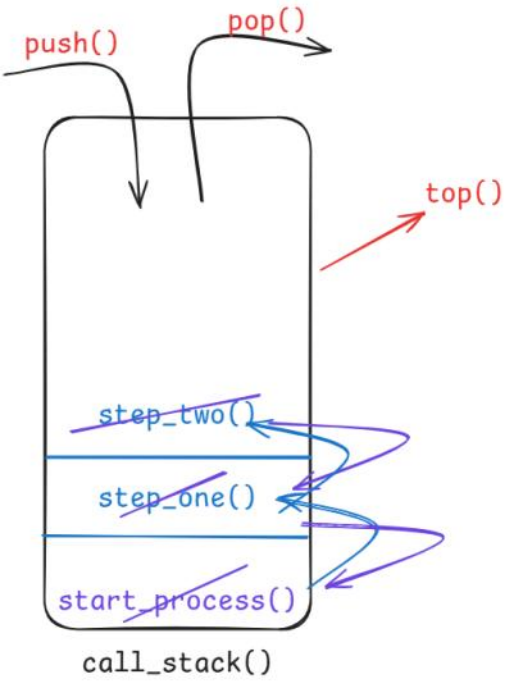


Console:

```
200
101
21
51
# NameError
101
```

```python
# Nested Function with Internal Call -> Call Stack()
def start_process():
    print("Starting the process..")
    step_one()
    print("Main Process Complete.")

def step_one():
    print("Starting Step One")
    step_two()
    print("Step One Complete")

def step_two():
    print("Starting Step Two")
    print("Performing the Final Operations....")
    print("Step Two Complete")

start_process()
```

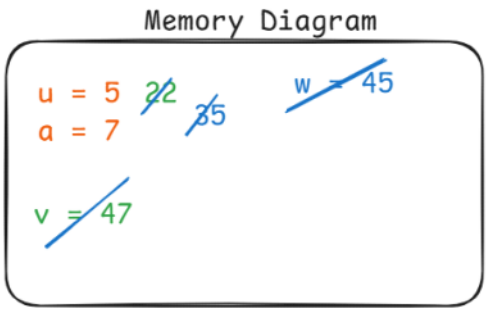
```
Starting the process..
Starting Step One
Starting Step Two
Performing the Final Operations....
Step Two Complete
Step One Complete
Main Process Complete.
```

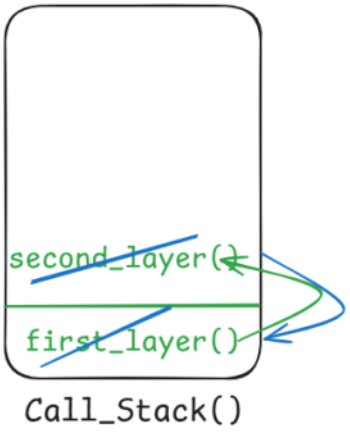


LIFO
Last In First Out

```python
# Shadowing in Nested Scope:
u = 5
a = 7
def first_layer():
    u = a + 15 # 22
    v = u + 25 # 47
    def second_layer():
        u = 35
        w = 45
        print("In Second Layer:")
        print(u) # 35
        print(v) # 47
        print(w) # 45
    print("In First Layer: ")
    print(u) # 22
    print(v) # 47
    second_layer()
first_layer()
print(u) # 5
# print(v) # NameError
```

enclosing scope





Console:

```
In First Layer:
22
47
In Second Layer:
35
47
45
5
```

```python
def is_armstrong(val):
    str_val = str(val) # '153'
    num_digits = len(str_val) # 3
    sum_of_power = 0
    for digit in str_val: # ['1','5','3']
        sum_of_power += int(digit) ** num_digits
    return sum_of_power == val # Boolean Return

is_armstrong(153)
```

```
True
```

```python
is_armstrong(371)
```

```
True
```

```python
is_armstrong(9474)
```

```
True
```

```python
is_armstrong(21) # 2^2 + 1^2 = 4+1 = 5
```

```
False
```

```python
def is_armstrong(val):
    str_val = str(val) # '153'
    num_digits = len(str_val) # 3
    sum_of_power = 0
    for digit in str_val: # ['1','5','3']
        sum_of_power += int(digit) ** num_digits
    return sum_of_power

val = int(input("Enter the value to check is_armstrong: "))
result = is_armstrong(val)
if result == val:
    print(f"{val} is a Armstrong.")
else:
    print(f"{val} is not a Armstrong.")
```

```
Enter the value to check is_armstrong:  153
153 is a Armstrong.
```

```python
# GCD [Greatest Common Divisor] # |-11| = 11
def gcd(x,y):
    while y!=0 :
        x,y = y , x%y
    return abs(x)
gcd(12,18)
```

```
6
```

```python
def gcd(x,y):
    while y!=0 :
        x,y = y , x%y
    return abs(x)
gcd(24,32) # 8
```

```
8
```

```python
# LCM [Least Common Multiple]
def lcm(x,y):
    return abs(x * y) // gcd(x,y)

lcm(12,18) # 36
```

```
36
```

Passing a Function as an Argument:

In Python Functions are like `First Class Citizen` :

1. Assigned to Variables.
2. Passes as arguments to other functions.
3. Returned from other Functions.

```python
def transform_list(func , items): # func , items -> iterables
    return [func(item) for item in items] # list Comprehension f(x)

def apply_discount(price):
    return round(price * 0.90, 2) # 10% Discount -> 0.90

def add_tax(price):
    return round(price * 1.05, 2) # 5% Standard Tax

prices = [100,250,499,50,120]
discounted = transform_list(apply_discount , prices)
taxed = transform_list(add_tax , discounted)

print("Original Price: " , prices)
print("Discounted Price: " , discounted)
print("Discounted Price with taxes: " , taxed)
```

```
Original Price:  [100, 250, 499, 50, 120]
Discounted Price:  [90.0, 225.0, 449.1, 45.0, 108.0]
Discounted Price with taxes:  [94.5, 236.25, 471.56, 47.25, 113.4]
```

```python
def transform_list(func , items): # func , items -> iterables
    return_list = []
    for item in items:
        val = func(item)
        return_list.append(val)
    return return_list

def apply_discount(price):
    return round(price * 0.90, 2) # 10% Discount -> 0.90

def add_tax(price):
    return round(price * 1.05, 2) # 5% Standard Tax

prices = [100,250,499,50,120]
discounted = transform_list(apply_discount , prices)
taxed = transform_list(add_tax , discounted)

print("Original Price: " , prices)
print("Discounted Price: " , discounted)
print("Discounted Price with taxes: " , taxed)
```

```
Original Price:  [100, 250, 499, 50, 120]
Discounted Price:  [90.0, 225.0, 449.1, 45.0, 108.0]
Discounted Price with taxes:  [94.5, 236.25, 471.56, 47.25, 113.4]
```

```python
# Returning Sum and Product
def sum_and_product(x,y):
    return x+y , x*y

_sum , _prod = sum_and_product(11,7)
print(_sum) # 18
print(_prod) # 77
```

```
18
77
```

```python
# Returning Sum and Product
def arithematic_operation(x,y):
    return x+y , x*y , x-y , x//y

_sum , _prod, _sub , _div = arithematic_operation(11,7)
print(_sum) # 18
print(_prod) # 77
print(_sub) # 4
print(_div) # 1
```

```
18
77
4
1
```

```python
# Global Scope Vs Local Scope
x = 10 # 'global'
def greet():
    y = 15 # 'local'
    print(x) # 10
    print(y) # 15
greet()
```

```
10
15
```

```python
# Global Scope Vs Local Scope
x = 10 # 'global'
def greet():
    y = 15 # 'local'
    print(x) # 10
    print(y) # 15
greet()
print(x) # 10
# print(y) # NameError: name 'y' is not defined
```

```
10
15
10
```

```python
# Local Scope:
def greet():
    m = 10 # 'local'
    n = 15 # 'local'
    print(m) # 10
    print(n) # 15
greet()
# print(m) # NameError
# print(n) # NameError
```

```
10
15
```

```python
gesture = 'Happy' # 'global'
def how_you_feel():
    gesture = "worried" # 'local'
    print(f"I'm feeling {gesture}") # 'Worried'

print(f"I'm feeling {gesture}") # 'Happy'
how_you_feel() # 'Worried'
print(f"I'm feeling {gesture}") # 'Happy'
```

```
I'm feeling Happy
I'm feeling worried
I'm feeling Happy
```

```python
# 'global' keyword
i = 10 # 'global'
def greet():
    global j # 'local' -> 'global'
    j = 15
    print(i) # 10
    print(j) # 15

greet()
print(i) # 10
print(j) # 15
```

```
10
15
10
15
```

```python
# Nested Function [LEGB]
p = 11 # 'global'
def outer_fx():
    q = 21 # 'enclosing'
    def inner_fx():
        r = 51 # 'local'
        s = 101 # 'local'
        print(p) # 11
        print(q) # 21
        print(r) # 51
        print(s) # 101
        print(len(['a','b','c','d',1,False,'Coding'])) # 7 ['Built In']
    print(p) # 11
    print(q) # 21
    inner_fx()
outer_fx()
print(p) # 11
```

```
11
21
11
21
51
101
7
11
```

```python
# Nested Function [LEGB]
p = 11 # 'global'
def outer_fx():
    q = 21 # 'enclosing'
    def inner_fx():
        r = 51 # 'local'
        s = 101 # 'local'
        print(p) # 11
        print(q) # 21
        print(r) # 51
        print(s) # 101
        print(len(['a','b','c','d',1,False,'Coding'])) # 7 ['Built In']
    print(p) # 11
    print(q) # 21
    # inner_fx()
outer_fx()
print(p) # 11
```

```
11
21
11
```

```python
a = 11
b = 21
def outer_fx():
    c = 51
    d = 77
    def inner_fx():
        e = 99
        global f
        f = 101
        print(e+f) # 200
    inner_fx()
    print(f) # 101
    print(b) # 21
    print(c) # 51
    # print(e) # NameError
outer_fx()
print(f) # 101
```

```
200
101
21
51
101
```

```python
# Shadowing in Nested Scope:
u = 5
def first_layer():
    u = 15
    v = 25
    def second_layer():
        u = 35
        w = 45
        print("In Second Layer:")
        print(u) # 35
        print(v) # 25
        print(w) # 45
    print("In First Layer: ")
    print(u) # 15
    print(v) # 25
    second_layer()
first_layer()
print(u) # 5
# print(v) # NameError
```

```
In First Layer:
15
25
In Second Layer:
35
25
45
5
```

```python
# Shadowing in Nested Scope:
u = 5
def first_layer():
    u = 15
    v = 25
    def second_layer():
        u = 35
        w = 45
        print("In Second Layer:")
        print(u) # 35
        print(v) # 25
        print(w) # 45
    print("In First Layer: ")
    print(u) # 15
    print(v) # 25
    second_layer()
first_layer()
print(u) # 5
# print(v) # NameError
```

```
In First Layer:
15
25
In Second Layer:
35
25
45
5
```

```python
# Shadowing in Nested Scope:
u = 5
a = 7
def first_layer():
    u = a + 15 # 22
    v = 25
    def second_layer():
        u = 35
        w = 45
        print("In Second Layer:")
        print(u) # 35
        print(v) # 25
        print(w) # 45
    print("In First Layer: ")
    print(u) # 22
    print(v) # 25
    second_layer()
first_layer()
print(u) # 5
# print(v) # NameError
```

```
In First Layer:
22
25
In Second Layer:
35
25
45
5
```

```python
# Shadowing in Nested Scope:
u = 5
a = 7
def first_layer():
    u = a + 15 # 22
    v = u + 25 # 47
    def second_layer():
        u = 35
        w = 45
        print("In Second Layer:")
        print(u) # 35
        print(v) # 47
        print(w) # 45
    print("In First Layer: ")
    print(u) # 22
    print(v) # 47
    second_layer()
first_layer()
print(u) # 5
# print(v) # NameError
```

```
In First Layer:
22
47
In Second Layer:
35
47
45
5
```

```python
# Nested Function with Internal Call -> Call Stack()
def start_process():
    print("Starting the process..")
    step_one()
    print("Main Process Complete.")

def step_one():
    print("Starting Step One")
    step_two()
    print("Step One Complete")

def step_two():
    print("Starting Step Two")
    print("Performing the Final Operations....")
    print("Step Two Complete")

start_process()
```

```
Starting the process..
Starting Step One
Starting Step Two
Performing the Final Operations....
Step Two Complete
Step One Complete
Main Process Complete.
```