Imam Abdulrahman Bin Faisal University
Deanship of Preparatory Year and Supporting Studies

Basics of Programming through Python **Conditional Statements**

Introduction to programming

COMP102

Term 3-2022-2023

Python Conditional Statements

Learning outcomes

- Understand the concept and usage of selection structure and conditional statements.
- Know various types of conditional statements available in Python(if, elif, Nested if).
- Handling an exception using try, except else and finally.
- Analyze the problem, decide and evaluate conditions.

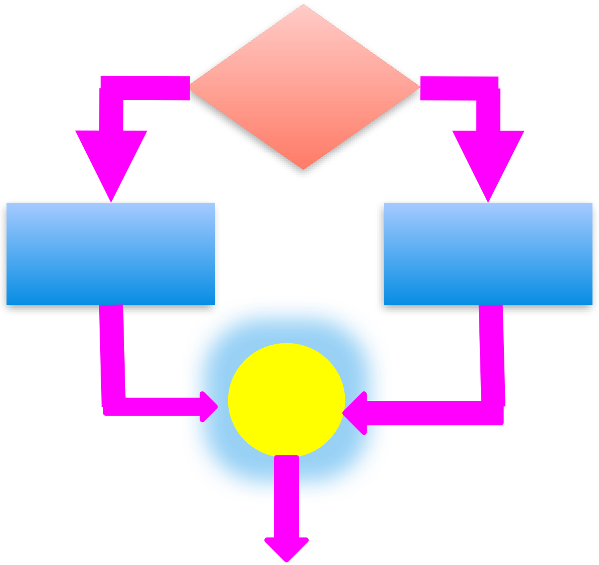
Types of control structures

A Structured programming is an important feature of a programming language which comprises following logical structure:

1. **SEQUENCE**
2. **SELECTION**
3. **ITERATION OR LOOPING**

Selection Structure

A selection statement causes the program control to be transferred to a specific flow based upon whether a certain condition is true or not.



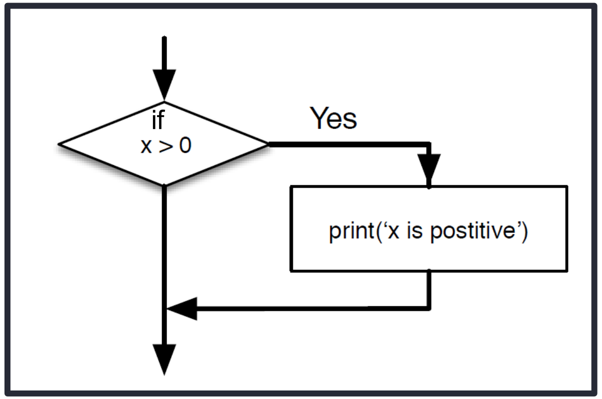
Introduction to conditional statements

- Conditional statements are used to control the flow of the program.
- Conditional statements in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false
- if, elif and else are the conditional statements in Python.

Conditional Exécution: if statement

- *Conditional statements* check conditions and change the behavior of the program accordingly.
- The simplest form is the ***if statement***: (one selection statement)

```
if x > 0 :
    print('x is positive')
```



If Logic

if--else statement

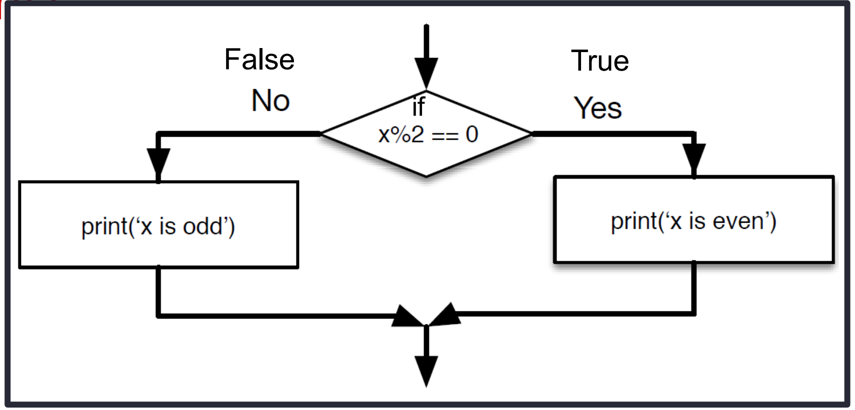
- An “if statement” is written by using the **if** keyword.
- Python if statement is used for decision-making operations.
- It Contains a body of code which runs only when the condition given in the if statement is **True**. If the condition is **False**, then the optional **else** statement runs which contains some code for the **else** condition.
- **Syntax of iF-else statement:**

```
if condition:
        statement_1_True
else:
```

Alternative Execution: if-else

- There are two possibilities and the condition determines which one gets executed. (two-way selection statement)

```
if x%2 == 0 :
    print('x is even’)
else :
    print('x is odd')
```



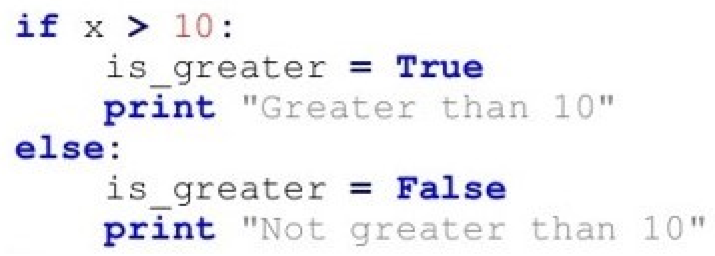
If-Then-Else Logic

If---else statements

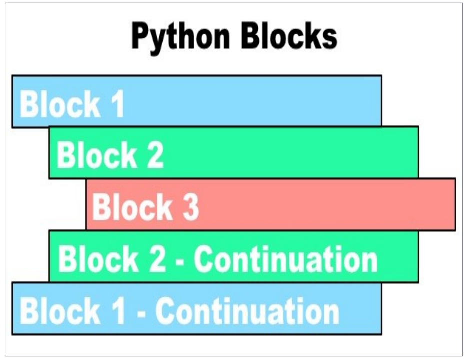
- There are a few important items to remember about **if statements:**
- The colon(:) is important and essential.
- The header of the compound statement(i.e. if statement) is isolated from the body.
- All rows **indented** after the colon will be executed whenever the Boolean expression is valid.

Python Blocks

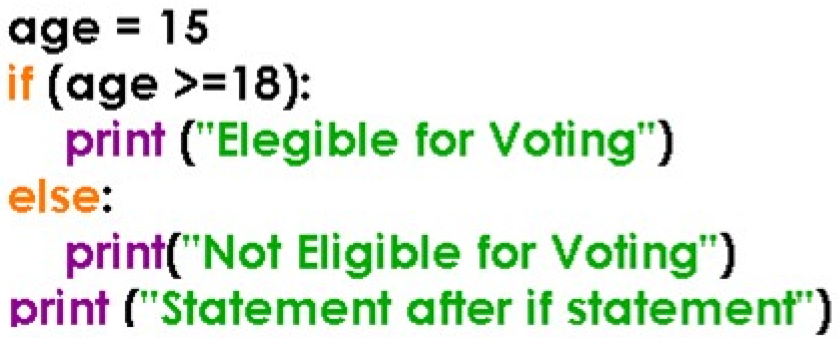
- Python uses indentation for blocks and nested blocks.
- Code Blocks
  - A code block is a set of statements that will be executed together, one after the other
  - If statements, for loops, while loops, functions,
  - Example:



```
if x > 10:
    is_greater = True
    print "Greater than 10"
else:
    is_greater = False
    print "Not greater than 10"
```

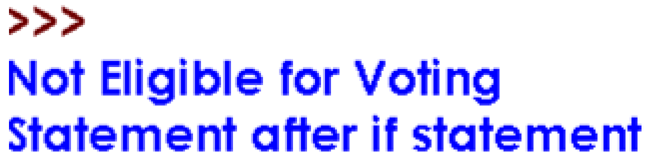


if---else: Example 1



```
age = 15
if (age >=18):
    print ("Elegible for Voting")
else:
    print("Not Eligible for Voting")
print ("Statement after if statement")
```

OUTPUT



```
>>>
Not Eligible for Voting
Statement after if statement
```

if---else: Example 2

- **Find the output of this code?**

```
A=15
B=20
If B>A:
    print(“B is greater than A”)
Else:
    print(“A is greater than B”)
```

OUTPUT

```
B is greater than A
```

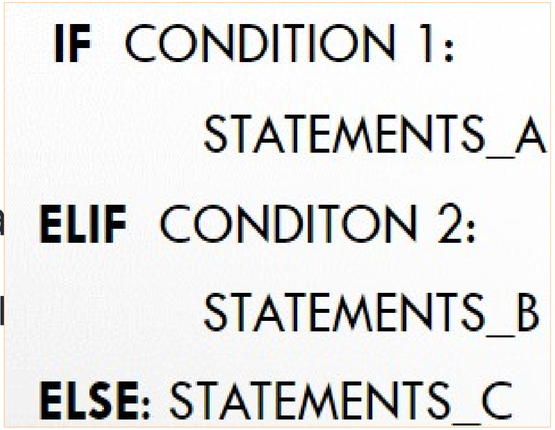
if …. elif Statement

- The **elif** keyword is python way of saying :”if the previous conditions were not true, then try this condition.
- **elif** is abbreviation of **else if**.
- There is no limit of the number of **elif** sta

  but only a single final else is allowed ar

must

  be the last branch in the statement.

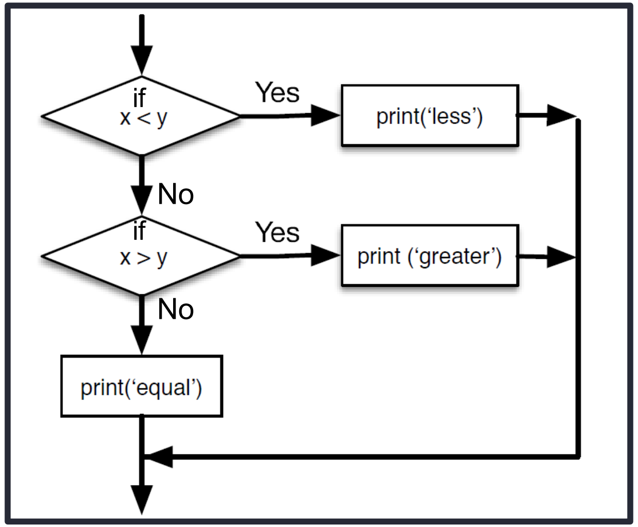


```
IF  CONDITION 1:
        STATEMENTS_A
ELIF  CONDITON 2:
        STATEMENTS_B
ELSE: STATEMENTS_C
```

Chained Conditionals

- Sometimes, there are more than two possibilities, and we need more than two branches. (Multiple-Way)

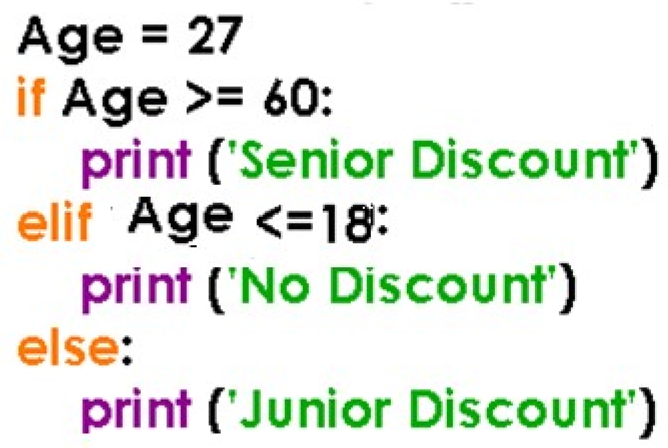
```
if x < y:
   print('x is less than y')
elif x > y:
   print('x is greater than y')
else:
   print('x and y are equal')
```



If-Then-ElseIf Logic

If---elif: Example 1

Find the output of this code:

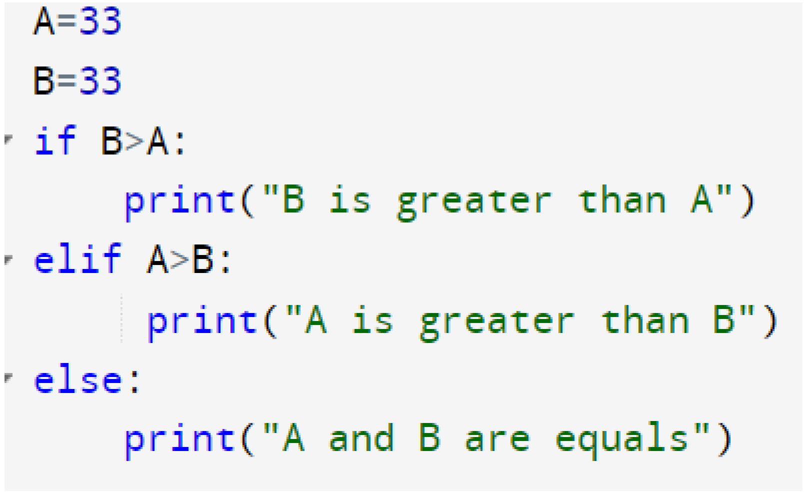


```
Age = 27
if Age >= 60:
    print ('Senior Discount')
elif  Age <=18:
    print ('No Discount')
else:
    print ('Junior Discount')
```

Output: Junior Discount

If—elif: Example 2

Find the output of this code:

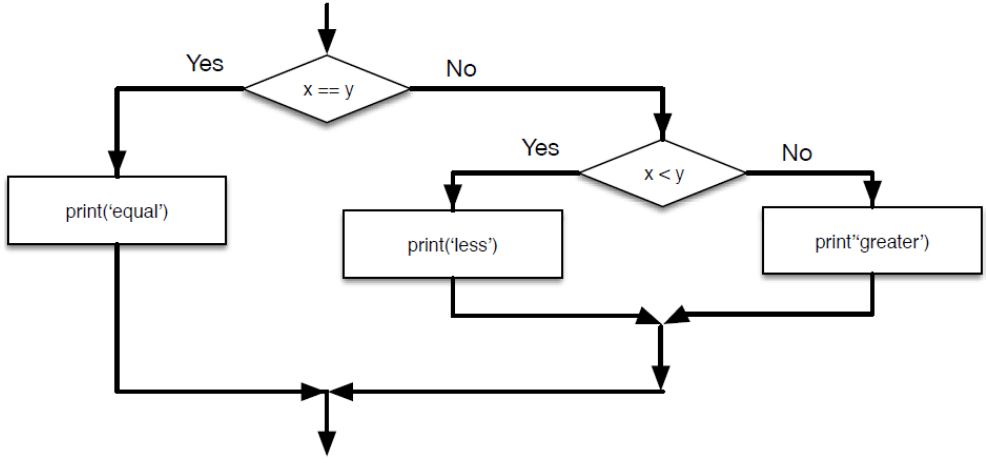


```
A=33
B=33
if B>A:
    print("B is greater than A")
elif A>B:
     print("A is greater than B")
else:
    print("A and B are equals")
```

Nested Conditionals

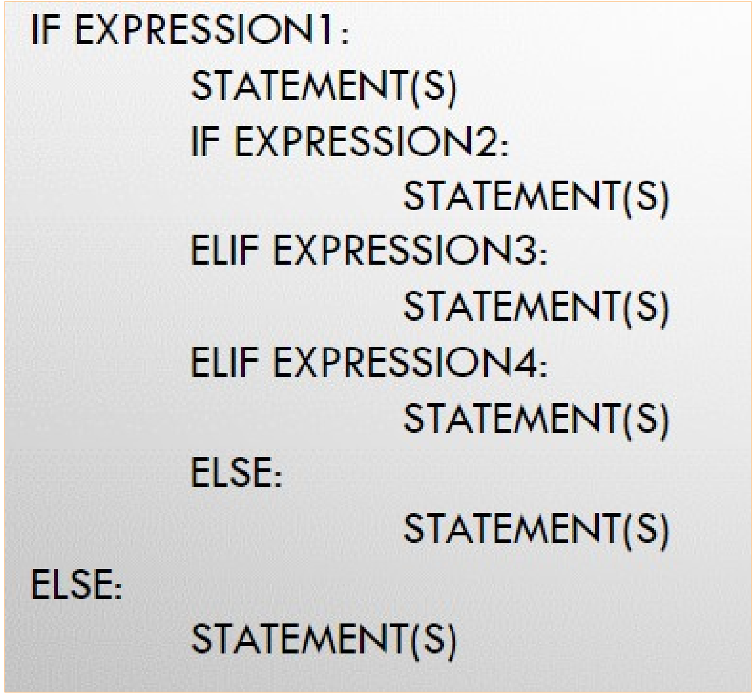
- One conditional can also be nested within another:

```
if x == y:
   print('x and y are equal')
else:
    if x < y:
        print('x is less than y’)
    else:
        print('x is greater than
y')
```



Nested If Statements

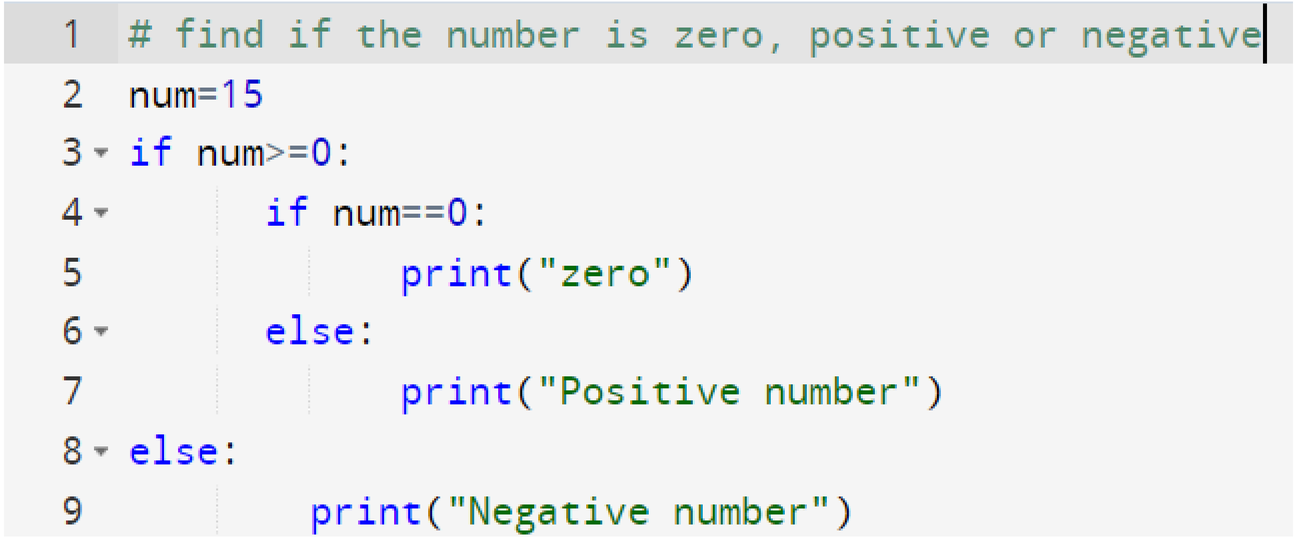
Nested If Statement---syntax



```
IF EXPRESSION1:
        STATEMENT(S)
        IF EXPRESSION2:
                STATEMENT(S)
        ELIF EXPRESSION3:
                STATEMENT(S)
        ELIF EXPRESSION4:
                STATEMENT(S)
        ELSE:
                STATEMENT(S)
ELSE:
        STATEMENT(S)
```

Nested if…Example

- Find the output of the code below.
- Rewrite the code using if—elif statement



```python
# find if the number is zero, positive or negative
num=15
if num>=0:
        if num==0:
                print("zero")
        else:
                print("Positive number")
else:
    print("Negative number")
```

Comparison Operators

- **Boolean expressions** ask a question and produce a **Yes** or **No** result which we use to control program flow

- **Boolean expressions** using comparison operators to evaluate to **True / False** or **Yes / No**

- **Comparison operators** look at variables but **do not change** the variables

Python	**Meaning**
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	Greater than
!=	Not equal
is	The same as
is not	Not the same as

Remember: "=" is used for assignment.

Comparison Operators

```
x = 5
if x == 5 :
    print('Equals 5')
if x > 4 :
   print('Greater than 4')
if  x >= 5 :
    print('Greater than or Equals 5')
if x < 6 : print('Less than 6')
if x <= 5 :
    print('Less than or Equals 5')
if x != 6 :
    print('Not equal 6')
```

```
Equals 5
Greater than 4
Greater than or
Equals 5
Less than 6
Less than or Equals 5
Not equal 6
```

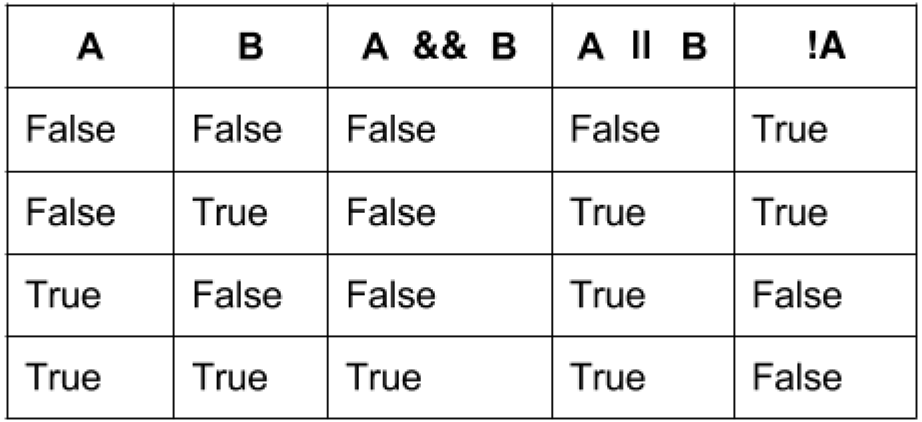
Logical Operators

- **There are three logical operators: and, or, and not.**
  - **x > 0 and x < 10** ☾ is **true** only **if** x is greater than 0 **and** less than 10.
  - **n%2 == 0 or n%3 == 0** ☾ is **true if either** one of the conditions is true, that is, if the number is divisible by 2 or 3
  - **not (x > y)** ☾ is **true** if **x > y is false**; that is, if x is less than or equal to y.

Any **nonzero** number is interpreted as **“True.”**

17 and True ☾ **True**

Truth table



A	B	A && B	A || B	!A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Python Exceptions Handling

- Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them:
  - **Exception Handling**
  - **Assertions**

What is Exception?

- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it can't cope with, it raises an exception. An exception is a Pytho
object that represents an error

Handling an exception

- If you have some *suspicious* code that may raise an **exception**, you can defend your program by placing the suspicious code in a **try:** block.
- After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax:

```
try:
    You do your
    operations here;
    ......................
except Exception I:
    If there is
    Exception, then
    execute this block.
```

Catching exceptions using try and except

- Here is a sample program to convert a Fahrenheit temperature to a Celsius temperature:

```
inp = input('Enter Fahrenheit
Temperature: ')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)
```

- If we execute this code and give it **invalid input**, it simply fails with an **unfriendly error** message

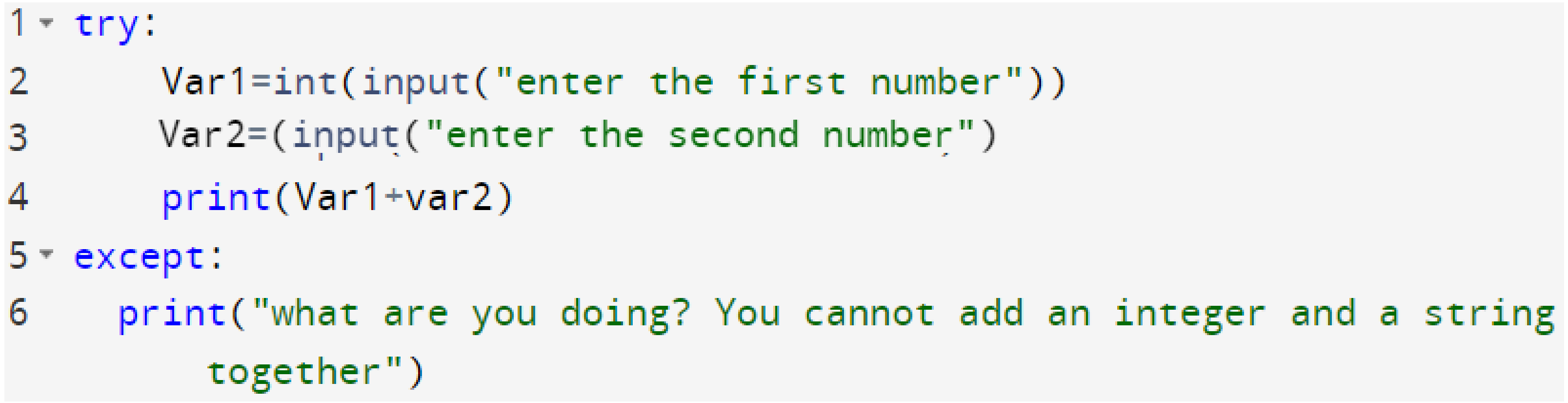
Catching exceptions using try and except

```
inp = input('Enter Fahrenheit
Temperature:')
try:
   fahr = float(inp)
   cel = (fahr - 32.0) * 5.0 / 9.0
   print(cel)
except:
    print('Please enter a number')
```

If an **exception** (**error**) occurs in the try block, Python jumps out of the try block and executes the sequence of statements in the **except** block.

- ❑ You can use try & except with any script code
- ❑ You can write try: at a beginning of your code or anywhere else
- ❑ You can write any script

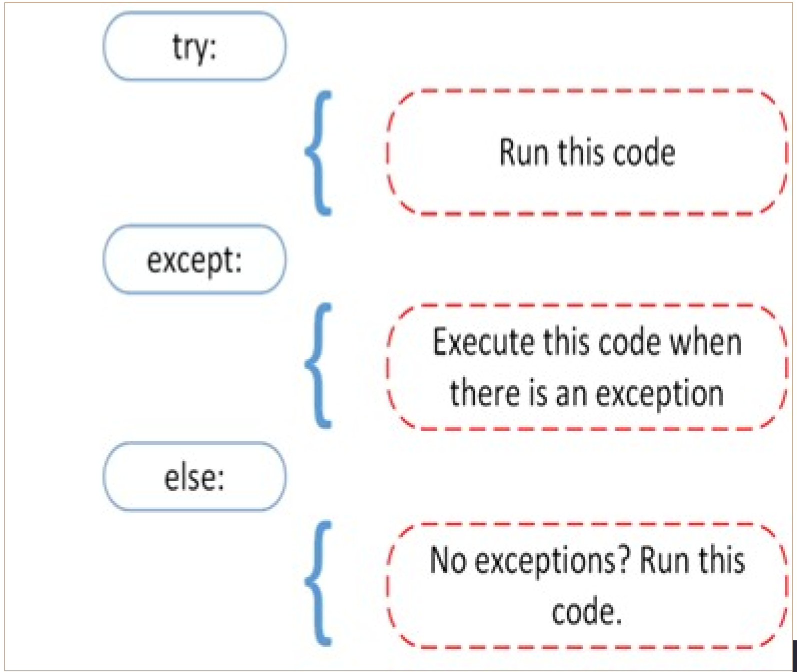
Try and Except: Find the output?



```
try:
    Var1=int(input("enter the first number"))
    Var2=(input("enter the second number")
    print(Var1+var2)
except:
  print("what are you doing? You cannot add an integer and a string
      together")
```

try except else

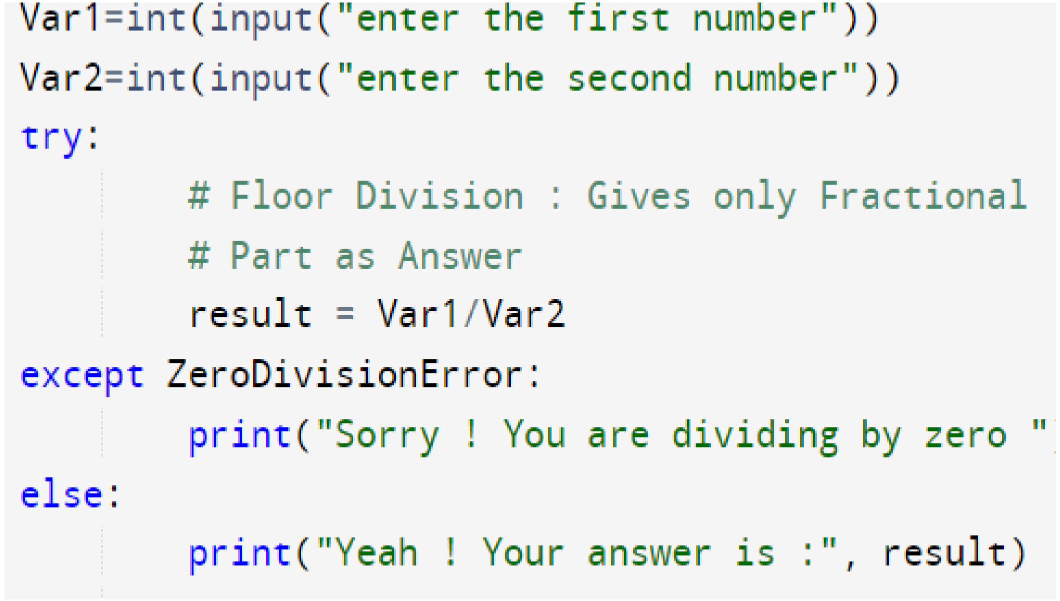
- After the except clause(s), you can include an **else-clause**. The code in the else-block executes if the code in the try: block does not raise an exception.
- The **else-block** is a good place for code that does not need the try: block's protection.


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.

Try and check

Find the output if:

- **Var 1=6 and Var**
- **Var 1=6 and Var 2=12**
- **Var 1=6 and Var**
- **Var 1=0 and Var**



```
Var1=int(input("enter the first number"))
Var2=int(input("enter the second number"))
try:
        # Floor Division : Gives only Fractional
        # Part as Answer
        result = Var1/Var2
except ZeroDivisionError:
        print("Sorry ! You are dividing by zero "
else:
        print("Yeah ! Your answer is :", result)
```

The Try-Finally Clause

- You can use a **finally:** block along with a **try:** block.
- The **finally block** is a place to put any code that must execute, whether the try-block raised an exception or not.
- Note that you can provide except clause(s), or a finally clause, **but not**

The syntax of the try-finally statement is this:

```
try:
    You do your operations here;
    ......................
    Due to any exception, this may be skipped.
finally:
    This would always be executed.
```

Try and check

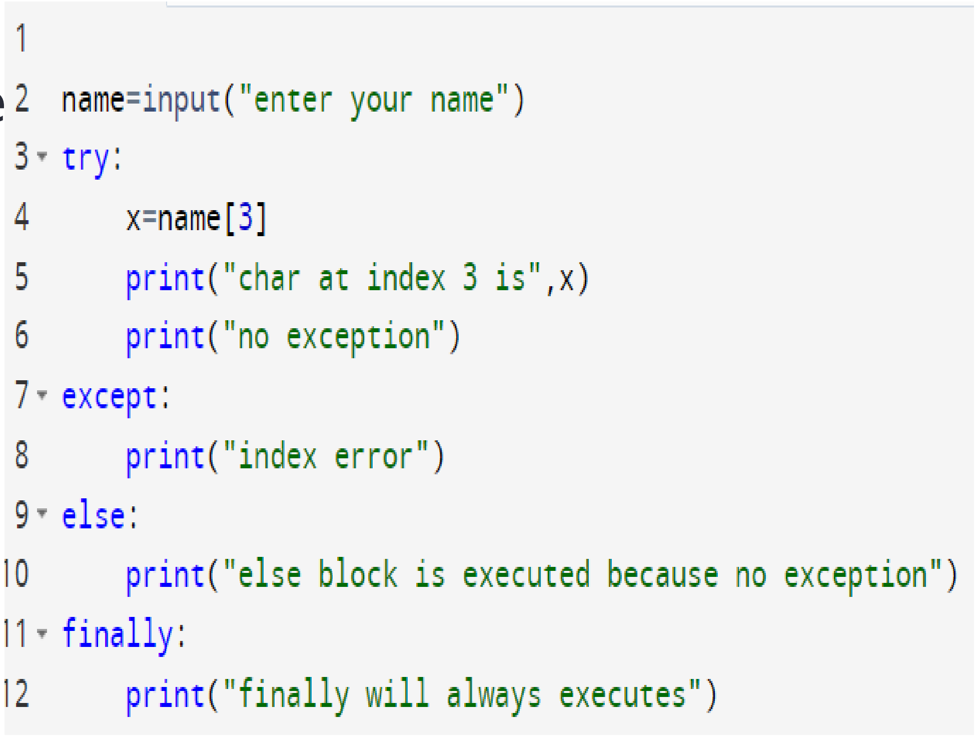
Find the output whe

name=Ahmed

name=Emy

name=Lee

name=Zaineb



```python

name=input("enter your name")
try:
    x=name[3]
    print("char at index 3 is",x)
    print("no exception")
except:
    print("index error")
else:
    print("else block is executed because no exception")
finally:
    print("finally will always executes")
```

Short-circuit evaluation of logical expressions

- When the **evaluation** of a **logical expression** stops because the overall value is **already known**, it is called ***short-circuiting.***

```
x = 6
y = 0
x >= 2 and (x/y) > 2
Error
```

```
x = 1
y = 0
x >= 2 and (x/y) > 2
False
```

y=0, which causes a runtime error (division by zero)

The first part of these expressions x >= 2 evaluated to False so the (x/y) was not ever executed

Short-circuit evaluation of logical expressions

- A **guard evaluation** can be strategically placed before the evaluation that might cause an error.

```
 x = 1
 y = 0
 x >= 2 and y != 0 and
(x/y) > 2
```

☾ **False**

```
 x = 6
 y = 0
x >= 2 and y != 0 and (x/y)
> 2
```

☾ **False**

```
 x = 6
 y = 0
x >= 2 and (x/y) > 2 and y !=
0
```

☾ **Error (without guard evaluation)**

Try and check 1

Rewrite your pay computation to give the employee 1.5 times the hourly rate for hours worked above 40 hours.

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

Try and check 2

Rewrite your pay program using try and except so that your program handles non-numeric input gracefully by printing a message and exiting the program. The following shows two executions of the program:

```
Enter Hours: 20
Enter Rate: nine
Error, please enter numeric input
Enter Hours: forty
Error, please enter numeric input
```

Try and check 3

Write a program to prompt for a score between 0.0 and 1.0. If the score is out of range, print an error message. If the score is between 0.0 and 1.0, print a grade using the following table:

>= 0.9	**A**
>= 0.8	**B**
>= 0.7	**C**
>= 0.6	**D**
< 0.6	**F**

Run the program repeatedly as follows to test the various different values for input.

```
Enter score: 0.95
A
Enter score:
perfect
Bad score
Enter score: 10.0
Bad score
Enter score: 0.75
C
Enter score: 0.5
F
```

Thanks!

Any questions?