UNIT - II

CONTENT OUTLINE

1. Using the If statement
2. Using the else Clause
3. Using the elif clause
4. Creating while Loops
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions

1. Using the If statement

- Introducing the Password Program
- Examining the if Statement
- Creating Conditions
- Understanding Comparison Operators
- Using Indentation to Create Blocks
- Building Your Own if Statement

Introducing the Password Program

```
# Password

# Demonstrates the if statement

        print("Welcome to System Security Inc.")

        print("-- where security is our middle name\n")

        password = input("Enter your password: ")

            if password == "secret": print("Access Granted")

        input("\n\nPress the enter key to exit."
```

Examining the if Statement

The key to program Password is the if statement:

```
if password == "secret":

print("Access Granted")
```

The if statement is pretty straightforward. You can probably figure out what's happening just by reading the code. I

Creating Conditions

using the if Statement

In Python, there are three forms of the if...else statement.

1. if statement
2. if...else statement
3. if...elif...else statement

Understanding Comparison Operators in if Statement

Python supports the usual logical conditions from mathematics:

Equals: `a == b`

Not Equals: `a != b`

Less than: `a < b`

Less than or equal to: `a <= b`

Greater than: `a > b`

Greater than or equal to: `a >= b`

Using Indentation to Create Blocks

Using Indentation to Create Blocks

Indentation refers to the spaces at the beginning of a code line.

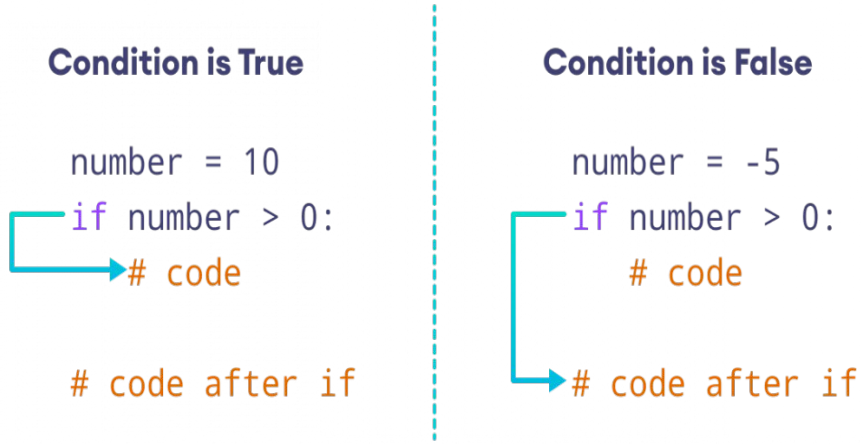
Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Building Your Own if Statement
Python if statement

The syntax of if statement in Python is:

```
if condition:
    # body of if statement
```




```
number = 10

# check if number is greater than 0

if number > 0:

print('Number is positive.')

print('The if statement is easy')
```

UNIT - II

CONTENT OUTLINE

1. Using the If statement
2. ***Using the else Clause***
3. Using the elif clause
4. Creating while Loops
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions

2. Using the else Clause

Using the else Clause

Introducing the Granted or Denied Program

Examining the else Clause

Python if else statement

The syntax of if-else statement in Python is:

```
if condition;
      # block of code if condition is True
else:
      # block of code if condition is False
```

Condition is True

```
number = 10
if number > 0:
    # code

else:
    # code

# code after if
```

Condition is False

```
number = -5
if number > 0:
    # code

else:
    # code

# code after if
```

```
number = 10

if number > 0:

    print('Positive number')

else:

    print('Negative number')

print('This statement is always executed')
```

UNIT - II

CONTENT OUTLINE

1. Using the If statement
2. Using the else Clause
3. ***Using the elif clause***
4. Creating while Loops
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions

3. Using the elif Clause

Using the elif Clause

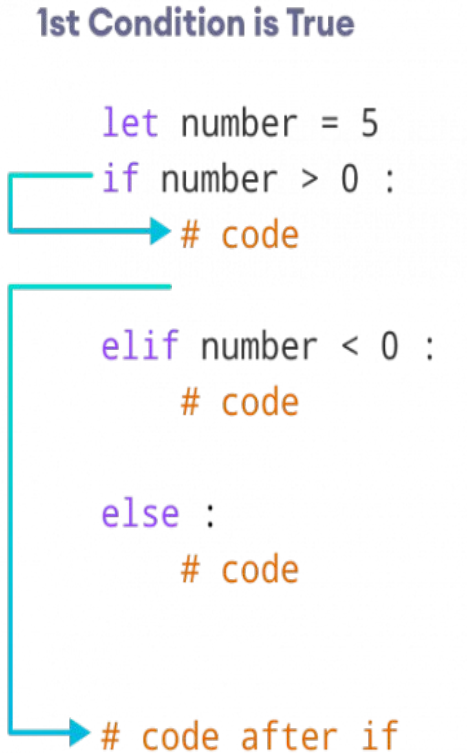
Introducing the Mood Computer Program

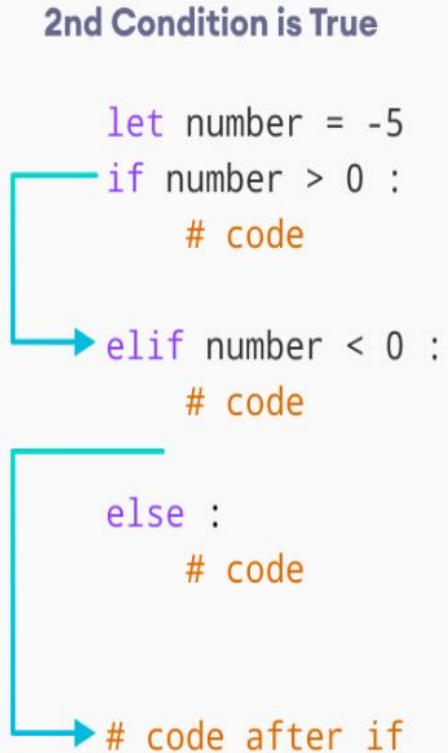
Examining the *elif* Clause

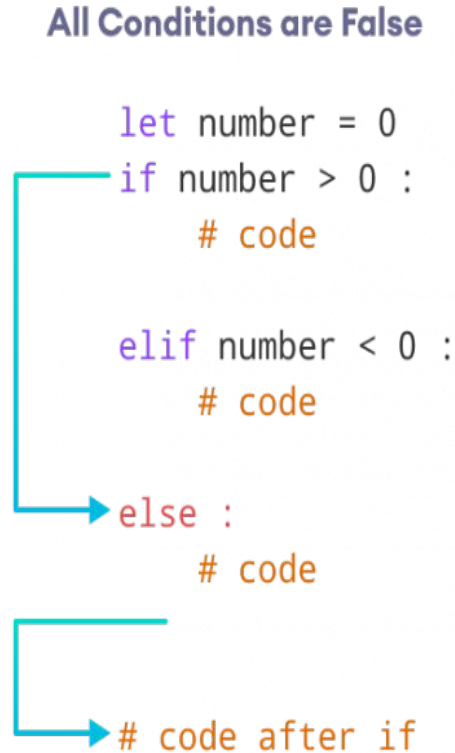
Python *if..elif..else* statement

The syntax of if...elif...else construct statement in Python is:

```
if condition1:
    # code block 1
elif condition2:
    # code block 2
else:
    # code block 3
```







Python *if..elif..else* statement

The syntax of if...elif...else construct *statement in Python is*:

```
if condition1:
        # code block 1
elif condition2:
         # code block 2
else:
        # code block 3
```

```
number = 0

if number > 0:

        print("Positive number")

elif number == 0:

        print('Zero’)

else:

        print('Negative number’)

print('This statement is always executed’)
```

Python *nested..if..* statement

The syntax of if...elif...else construct

statement in Python is:

```
if expression1:
        statement(s)
        if expression2:
                statement(s)
        else
                statement(s)
else
      if expression3:
                statement(s)
        else
                statement(s)
```

```
number = 5 # outer if statement


if (number >= 0):
        # inner if statement
if number == 0:
        print('Number is 0’)
# inner else statement
else:
        print('Number is positive’)
# outer else statement
else:
        print('Number is negative')
```

UNIT - II

CONTENT OUTLINE

1. Using the If statement
2. Using the else Clause
3. Using the elif clause
4. ***Creating while Loops***
5. Avoiding Infinite Loops
6. Creating Intentional infinite Loops
7. Using Compound Conditions

4. Creating while Loops

- Creating while Loops
- Introducing the Three-Year-Old Simulator Program
- Examining the while Loop
- Initializing the Sentry Variable
- Checking the Sentry Variable
- Updating the Sentry Variable

Introducing the Three-Year-Old Simulator Program

```
# Three Year-Old Simulator
# Demonstrates the while loop

print("\t Welcome to the 'Three-Year-Old Simulator'\n")
print("This program simulates a conversation with a three-year-old child.")
print("Try to stop the madness.\n")
response = ""
while response != "Because.":
        response = input("Why?\n")
print("Oh. Okay.")
```

Examining the while Loop

Examining the while Loop

```
while response != "Because.":
    response = input("Why?\n")
```

Initializing the Sentry Variable

- Often, while loops are controlled by a sentry variable, a variable used in the condition and compared to some other value or values.
- Like a human sentry, you can think of your sentry variable as a guard, helping form a barrier around the while loop's block.
- In the Three-YearOld Simulator program, the sentry variable is response.
- It's used in the condition and is compared to the string "Because." before the block is executed each time.
- It's important to initialize your sentry variable. Most of the time, sentry variables are initialized right before the loop itself.
- That's what I did with: response = "" If the sentry variable doesn't have a value when the condition is evaluated, your program will generate an error

Checking the Sentry Variable

Make sure that it's possible for the while condition to evaluate to True at some point;

Otherwise, the block will never run.

Take, for example, one minor change to the loop you've been working with

```
response = "Because."
while response != "Because.":
response = input("Why?\n")
```

Since response is equal to "Because." right before the loop, the block will never run.

Updating the Sentry Variable:

Updating the Sentry Variable:

Once you've established your condition, initialized your sentry variable,

and are sure that under some conditions the loop block will execute, you have yourself a working loop.

Next, make sure the loop will end at some point.

If you write a loop that never stops, you've created an infinite loop.

Welcome to the club. At one time or another, all programmers have accidentally created an infinite loop and watched their program get stuck doing something over and over.

Or they see their programs just plain freeze up.

Here's a simple example of an infinite loop:

```
counter = 0
while counter <= 10 print(counter)
```

UNIT - II

CONTENT OUTLINE

1. Using the If statement
2. Using the else Clause
3. Using the elif clause
4. Creating while Loops
5. ***Avoiding Infinite Loops***
6. Creating Intentional infinite Loops
7. Using Compound Conditions

5. Avoiding Infinite Loops

- Introducing the Losing Battle Program
- Tracing the Program
- Creating Conditions That Can Become False

Introducing the Losing Battle Program

Losing Battle # Demonstrates the dreaded infinite loop

```
print("Your hero unsheathes his sword for the last fight of his life.\n")
health = 10
trolls = 0
damage = 3
while  health != 0:
        trolls += 1
        health -= damage
print("Your hero swings and defeats an evil troll, " \ "but takes", damage, "damage points.\n")
print("Your hero fought valiantly and defeated", trolls, "trolls.")
```

Tracing the Program

Demonstrates the dreaded infinite loop

health	trolls	damage	health != 0
10	0	3	True
7	1	3	True
4	2	3	True
1	3	3	True
-2	4	3	True
-5	5	3	True
-7	6	3	True

Creating Conditions That Can Become False

The line with the condition just needs to become

while health > 0:

Now, if health becomes 0 or negative, the condition evaluates to False and the loop ends. To be sure, you can trace the program using this new condition:

health	trolls	damage	health != 0
10	0	3	True
7	1	3	True
4	2	3	True
1	3	3	True
-2	4	3	False

Treating Values as Conditions

- Introducing the Maitre D' Program
- Interpreting Any Value as True or False

Introducing the Maitre D' Program

Evaluating condition

```
print("Welcome to the Chateau D' Food")
print("It seems we are quite full this evening.\n")

money = int(input("How many dollars do you slip the Maitre D'? "))

if money:
    print("Ah, I am reminded of a table. Right this way.")
else:
    print("Please, sit. It may be a while.")

input("\n\nPress the enter key to exit.")
```

Interpreting Any Value as True or False

The new concept is demonstrated in the line:

```
if money:
```

Notice that money is not compared to any other value.
money is the condition.
When it comes to evaluating a number as a condition, **0 is False** and everything **else is True**.

So, the above line is equivalent to

```
if money != 0:
```

The first version is simpler, more elegant, and more intuitive. It reads more naturally and could be translated to "if there is money.

6. Creating Intentional Infinite Loops

- Introducing the Finicky Counter Program
- Using the break Statement to Exit a Loop
- Using the continue Statement to Jump Back to the Top of a Loop
- Understanding When to Use break and continue

Introducing the Finicky Counter Program

The Finicky Counter program counts from 1 to 10 using an intentional infinite loop. It's finicky because it doesn't like the number 5 and skips it. # Finicky Counter # Demonstrates the break and continue statements

```
count = 0
while True:
    count += 1
if count > 10:
    break
if count == 5:    (skips if 5)
    continue
print(count)
input("\n\nPress the enter key to exit.")
```

Using the break Statement to Exit a Loop

I set up the loop with:

```
while True:
```

This technically means that the loop will continue forever, unless there is an exit condition in the loop body. Luckily, I put one in:

end loop if count greater than 10

```
if count > 10:
    break
```

Since count is increased by 1 each time the loop body begins, it will eventually reach 11.

When it does, the break statement, which here means “break out of the loop,” is executed and the loop ends

Using the continue Statement to Jump Back to the Top of a Loop

Just before count is printed,

I included the lines:

```
# skip 5
    if count == 5:
    continue
```

The continue statement means “jump back to the top of the loop.”

Understanding When to Use break and continue

- You can use break and continue in any loop you create.
- They aren't just restricted for use in intentional infinite loops.
- But they should be used sparingly.
- Both break and continue make Branching, while Loops, and Program Planning

7. Using Compound Conditions and Operators

Arithmetic Operators

Assignment Operators

Comparison Operators

Identity Operators

Membership Operators

Bitwise Operators

Arithmetic Operator

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operator

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operator

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Identity Operator

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Membership Operator

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise Operator

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off