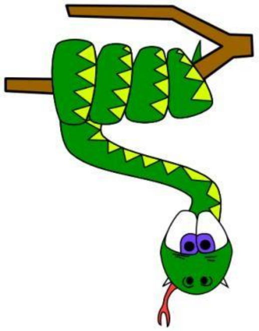


Introduction to Programming with Python

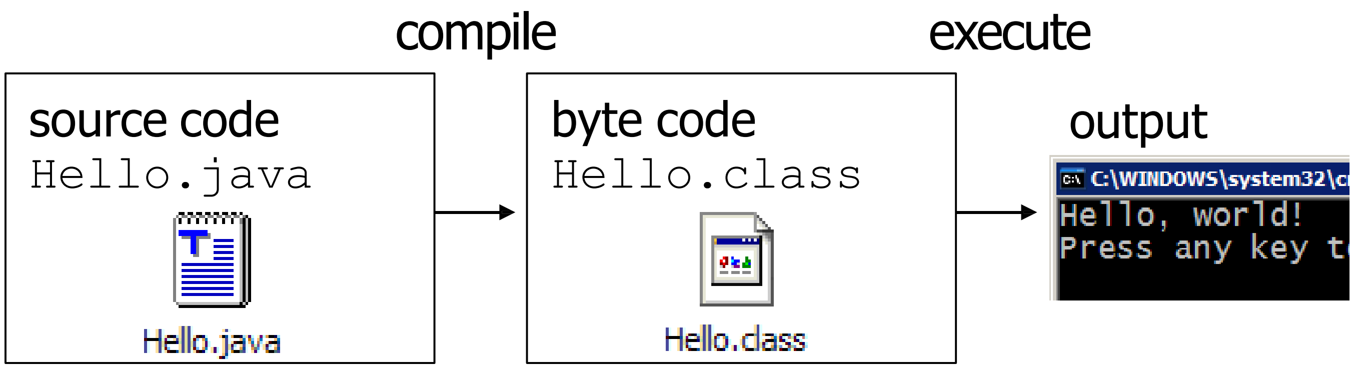
Programming basics

- **code** or **source code**: The sequence of instructions in a program
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.

console: The text box onto which output is printed.

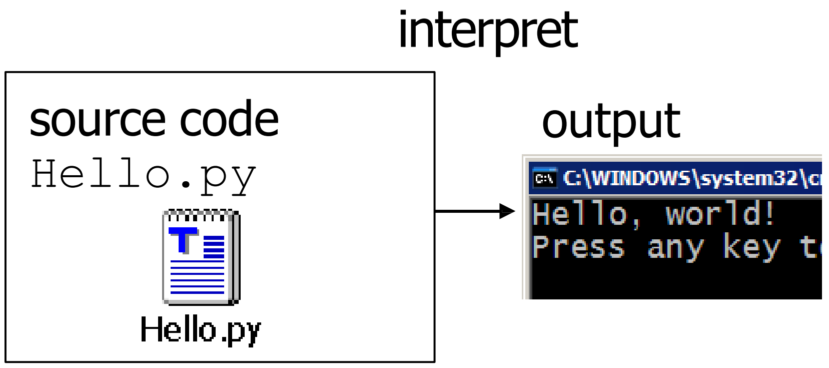
Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



```
Hello, world!
Press any key t
```

- Python is instead directly *interpreted* into machine instructions.



```
Hello, world!
Press any key t
```

Interpreted language

"Interpreted language" refers to a type of programming language where the code is executed line by line, or statement by statement, without the need for a separate compilation step.

Unlike compiled languages, where the source code is first translated into machine code before execution, interpreted languages do not require a separate compilation process.

The interpreter directly reads and executes the source code.

Why the name Python

The Python programming language is named after the Monty Python comedy troupe, not after a snake.

Guido van Rossum, the creator of Python, was a fan of the Monty Python's Flying Circus television series and named the language after it as a tribute.

About Python

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language.

Python has a simple syntax similar to the English language.

Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

It is simple and easy to learn and provides lots of high-level data structures.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.

- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.

- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Why learn Python?

Python provides many useful features to the programmer. These features make it the most popular and widely used language.

- **Easy to use and Learn:** Python has a simple and easy-to-understand syntax, unlike traditional languages like C, C++, Java, etc., making it easy for beginners to learn.

- **Open Source Language:** Python is open source and free to use, distribute and modify.

- **Interpreted Language:** Python does not require compilation, allowing rapid development and testing. It uses Interpreter instead of Compiler.

- **Wide Range of Libraries and Frameworks:**

Python has a vast collection of libraries and frameworks, such as NumPy, Pandas, Django, and Sklearn, that can be used to solve a wide range of problems.

Python is rich in libraries making it suitable for various tasks related to AI, ML and Deep learning.

Python has become popular among data scientists and machine learning engineers with libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and more.

- **Career Opportunities:** Python is a highly popular language in the job market. Learning Python can open up several career opportunities in data science, artificial intelligence, web development, and more.

- **Dynamic Memory Allocation:** Python automatically manages memory allocation, making it easier for developers to write complex programs without worrying about memory management.

- **Versatility:** Python is a universal language in various domains such as web development, machine learning, data analysis, scientific computing, and more.

The Python Interpreter

- Python is an interpreted language
- The interpreter provides an interactive environment to play with the language
- Results of expressions are printed on the screen

```
>>> 3 + 7
10
>>> 3 < 15
True
>>> 'print me'
'print me'
>>> print ('print
me’)
print me
```

Python Lines and Indentation

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

Unlike to C/C++, Python programming provides no braces to indicate blocks of code for class and function definitions or flow control.

Blocks of code are denoted by **line indentation.**

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

The number of spaces is up to a programmer, the most common use is four, but it has to be at least one.

Hence, in Python all the continuous lines indented with same number of spaces would form a block.

Variables

Python has no command for declaring a variable.

In Python, variables are created when you assign a value to it.

Python does not bind us to declare a variable before using it in the application.

It allows us to create a variable at the required time.

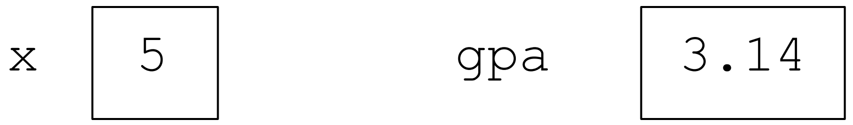
Variables

- **variable**: A named piece of memory that can store a value.
  - Usage:
    - Compute an expression's result,
    - store that result into a variable,
    - and use that variable later in the program.

- **assignment statement**: Stores a value into a variable.
  - Syntax:

    name = ***value***

  - Examples:
    ```
    x = 5
    gpa = 3.14
    ```



- A variable that has been given a value can be used in expressions.
  `x + 4` is `9`

One Value to Multiple Variables

you can assign the *same* value to multiple variables in one line:

```
x = y = z = "Hello"
print(x)
print(y)
print(z)
```

If there is a collection of values in a list, tuple etc. Python allows to extract the values into variables. This is called *unpacking*.

Python - Output Variables

In Python, print() function is often used to output variables

Using print() function, we can get multiple output variables, separated by a comma.

+ operator is used to output multiple variables.

In print function, when you try to combine a string and a number with the + operator, it will give you an error

print

- `print` : Produces text output on the console.

- Syntax:

  `print("`***Message***`)`

  `print(`***Expr)***

  - Prints the given text message or expression value on the console, and moves the cursor down to the next line.

  `print (`***Item1***`,` ***Item2***`,` ...`,` ***ItemN)***

- Examples:

```
print("Hello,world!")
age = 45
print ("You have", 65 - age, "years until retirement")
```

Output:

```
Hello, world!
You have 20 years until retirement
```

Expressions

- **expression**: A data value or set of operations to compute a value.

  Examples: `1 + 4 * 3`

  `42`

- Arithmetic operators we will use:

`+ - * /`	addition, subtraction/negation, multiplication, division
`%`	modulus, a.k.a. remainder
`**`	exponentiation

- **precedence**: Order in which operations are computed.
  - `* / % **` have a higher precedence than `+ -`

    `1 + 3 * 4` is `13`

  - Parentheses can be used to force a certain order of evaluation.

    `(1 + 3) * 4` is `16`

Integer division

- When we divide integers with `/` , the quotient is also an integer.

```
      3                    52
4 ) 14              27 ) 1425
    12                   135
     2                     75
                           54
                           21
```

  - More examples:
    - `35 / 5` is `7`
    - `84 / 10` is `8`
    - `156 / 100` is `1`

- The `%` operator computes the remainder from a division of integers.

```
      3                43
4 ) 14           5 ) 218
    12               20
     2                18
                      15
                       3
```

Real numbers

- Python can also manipulate real numbers.
  - Examples: `6.022` `-15.9997` `42.0` `2.143e17`

- The operators `+ - * / ( )` all work for real numbers.
  - The `/` produces an exact answer: `15.0 / 2.0` is **`7.5`**
  - The same rules of precedence also apply to real numbers:

- When integers and reals are mixed, the result is a real number.
  - Example: `1 / 2.0` is `0.5`

Math commands

- Python has useful commands (or called functions) for performing calculations.

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1*** `,` ***value2***`)`	larger of two values
`min(`***value1*** `,` ***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

- To use many of these commands, you must write the following at the top of your Python program:

```
from math import *
```

Numbers: Floating Point

- int(x) converts x to an integer
- float(x) converts x to a floating point
- The interpreter shows
  a lot of digits

```
>>> 1.23232
1.23232
>>> print 1.23232
1.23232
>>> int(2.0)
2
>>> float(2)
2.0
```

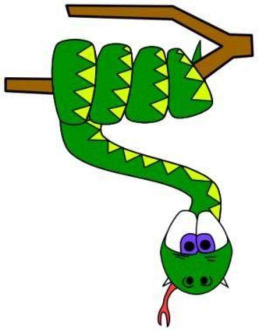
`input`

- `input` : Reads a number from user input.
  - You can assign (store) the result of `input` into a variable.
  - Example:

```
age = input("How old are you? ")
print ("Your age is", age)
print ("You have", 65 - age, "years until
retirement")
```

Output:

```
How old are you? 53
Your age is 53
You have 12 years until retirement
```

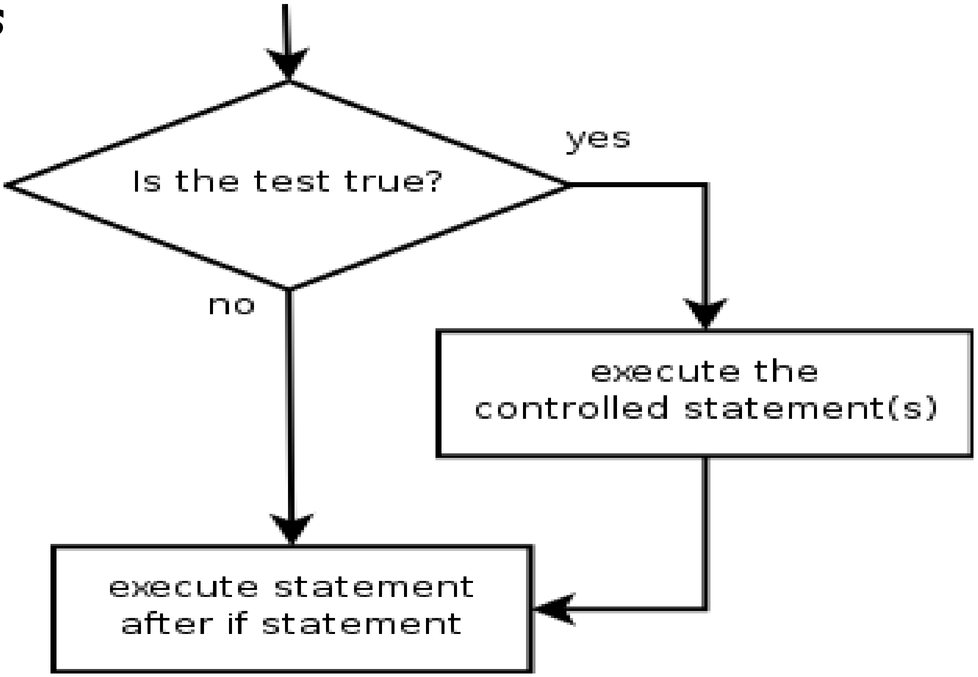


Repetition (loops) and Selection (if/else)

`if`

- **`if` statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
  - Syntax:

```
if condition:
    statements
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

- Example:

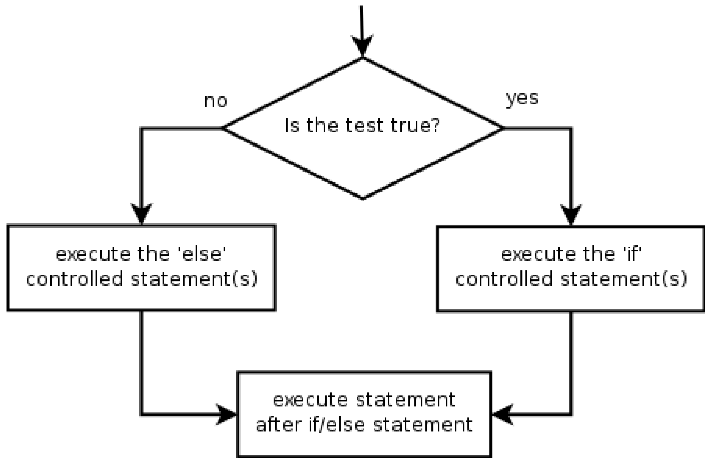
```
gpa = 3.4
if gpa > 2.0:
    print ("Your application is accepted.“)
```

```
num = int(input("enter the number:"))
# Here, we are taking an integer num and taking
input
if num%2 == 0:
    print("The Given number is an even number")
```

if/else

- **`if/else` statement**: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.
  - Syntax:

```
if condition:
    statements
else:
    statements
```



Is the test true?
no
yes
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

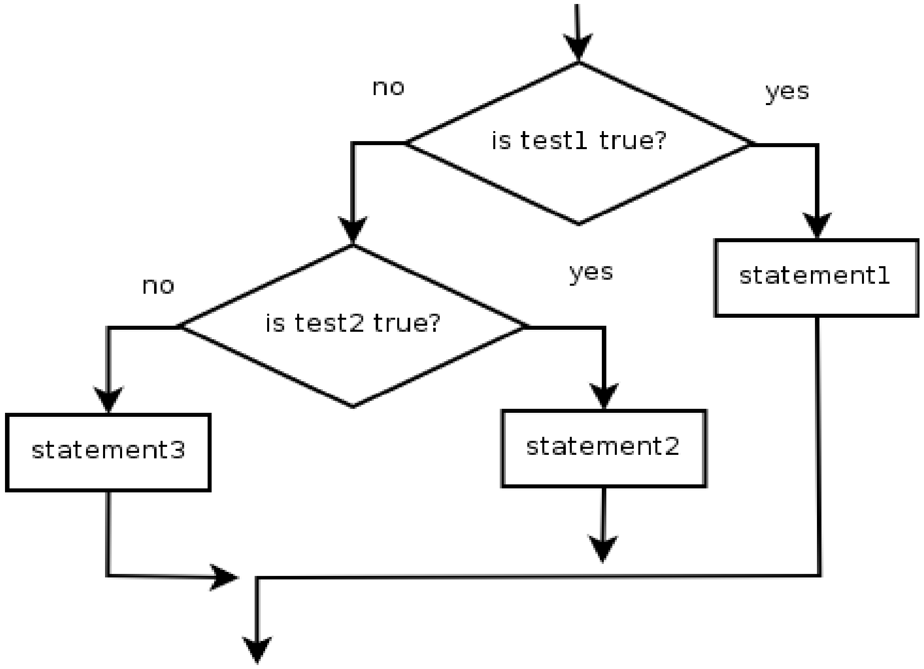
- Example:

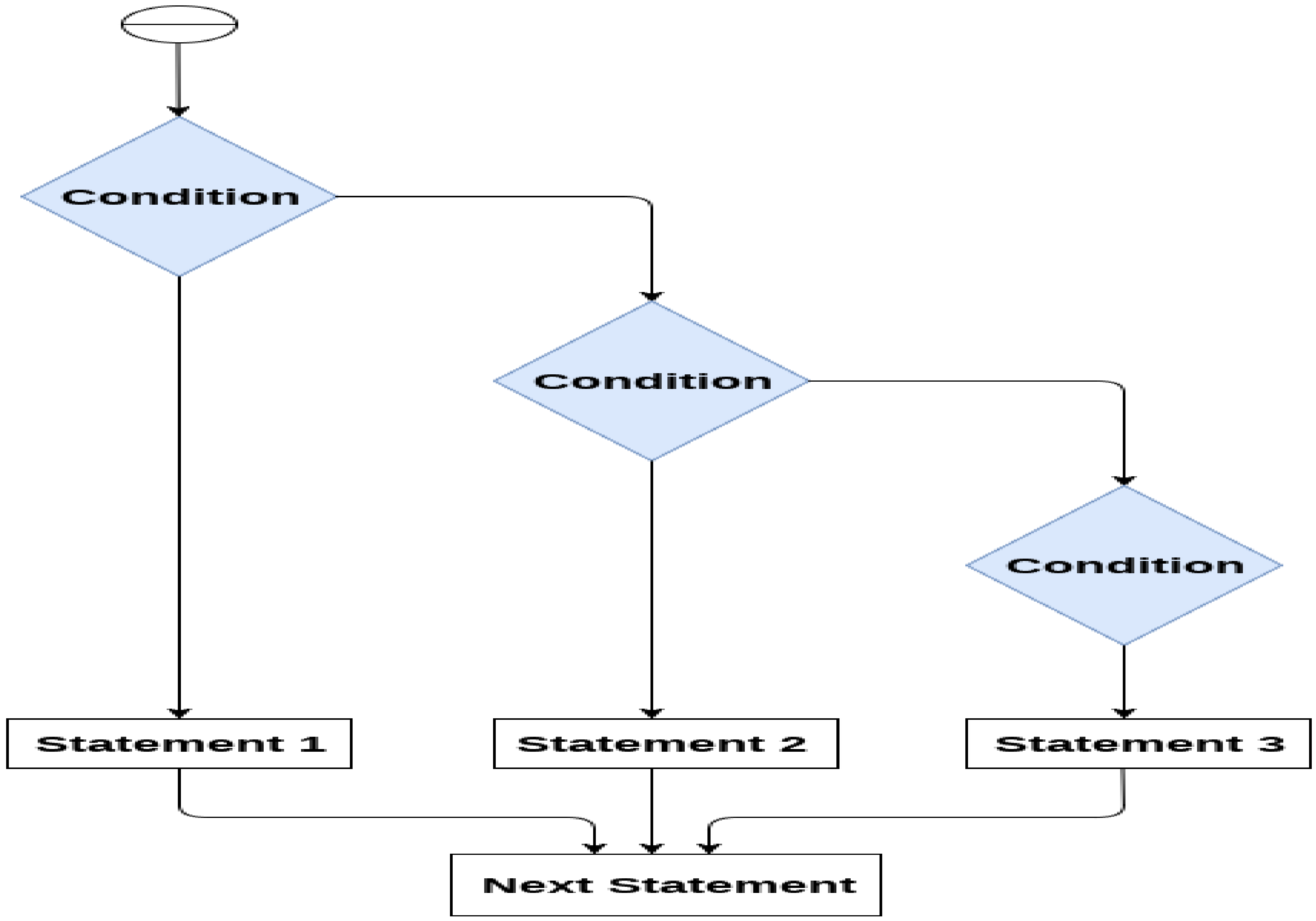
```
gpa = 1.4
if gpa > 2.0:
   print("Welcome to Data Analytics Course !“)
else:
   print ("Your application is denied.“)
```

`elif` ("else if"):

- Multiple conditions can be chained with `elif` ("else if"):

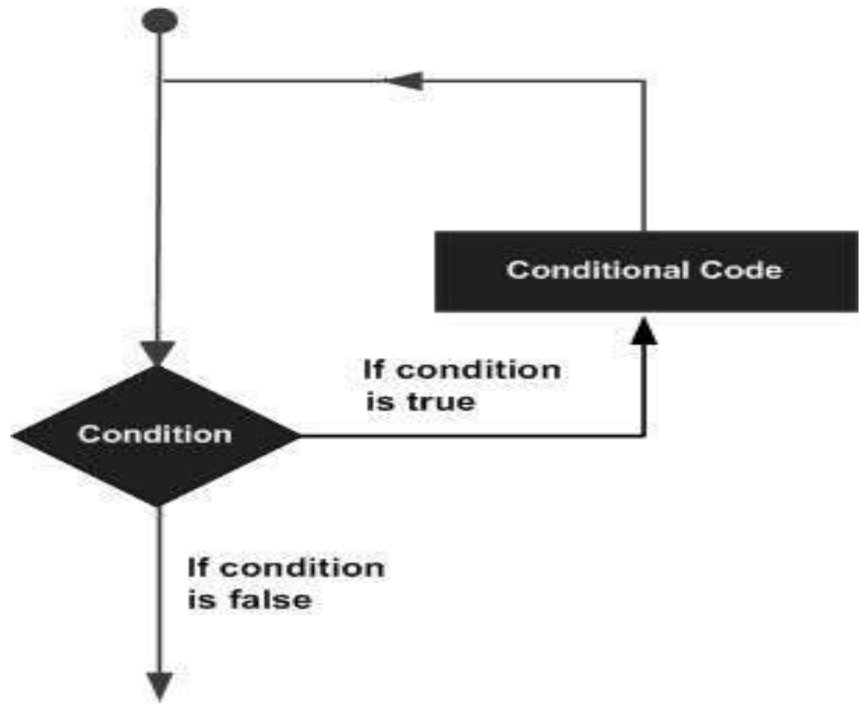
```
if expression 1:
  # block of statements

elif expression 2:
  # block of statements

elif expression 3:
  # block of statements

else:
  # block of statements
```




Condition
Condition
Condition
Statement 1
Statement 2
Statement 3
Next Statement

Python Loops

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −



`while`

- **`while` loop**: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)

- Syntax:

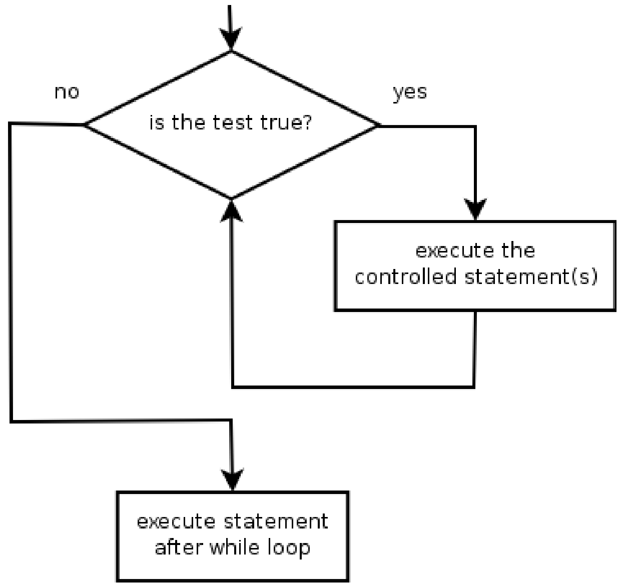
```
while condition:
      statements
```

- Example:

```
number = 1
while number < 200:
    print (number)
    number = number * 2
```

  - Output:

```
1 2 4 8 16 32 64 128
```


no
yes
is the test true?
execute the controlled statement(s)
execute statement after while loop

```
i = 1
while i < 10:
  print(i)
  i += 1
```

```
number= 1
while number< 200:
  print(number*number)
  number+= 1
```

The `for` loop

- **`for` loop**: Repeats a set of statements over a group of values.

  - Syntax:

```
for variableName   in groupOfValues:
    Statements
```

    - We indent the statements to be repeated with tabs or spaces

    .
    - ***variableName*** gives a name to each value, so you can refer to it in the ***statements***.

    - ***groupOfValues*** can be a range of integers, specified with the `range` function.

- Example:

```
for x in range(1, 6):
    print (x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

`range`

- The `range` function specifies a range of integers:
  - `range(`***start***`, `***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)

- It can also accept a third value specifying the change between values.
  - `range(`***start***`, `***stop***`, `***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

- Example:

```
for x in range(5, 0, -1):
    print x
print ("Blastoff!")
```

Output:

```
5
4
3
2
1
Blastoff!
```

for loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
fruits = [“grapes”, "apple", "banana", “mango"]
for i in fruits:
  print(i)
```

```
for i in “pineapple":
  print(i)
```

Print on same line with space between each element

The **end**=” “ is used to print in the same line with space after each element. It prints a space after each element in the same line.

```
for i in range(10):
    print(i, end=" ")
```

Cumulative loops

- Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a *cumulative sum*.

```
sum = 0
for i in range(1, 11):
    sum = sum + (i * i)
print ("sum of first 10 squares is", sum)

Output:
sum of first 10 squares is 385
```

- **Exercise:** Write a Python program that computes the factorial of an integer.

Logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

- **Exercise:** Write code to display and count the factors of a number.