Introduction to Python

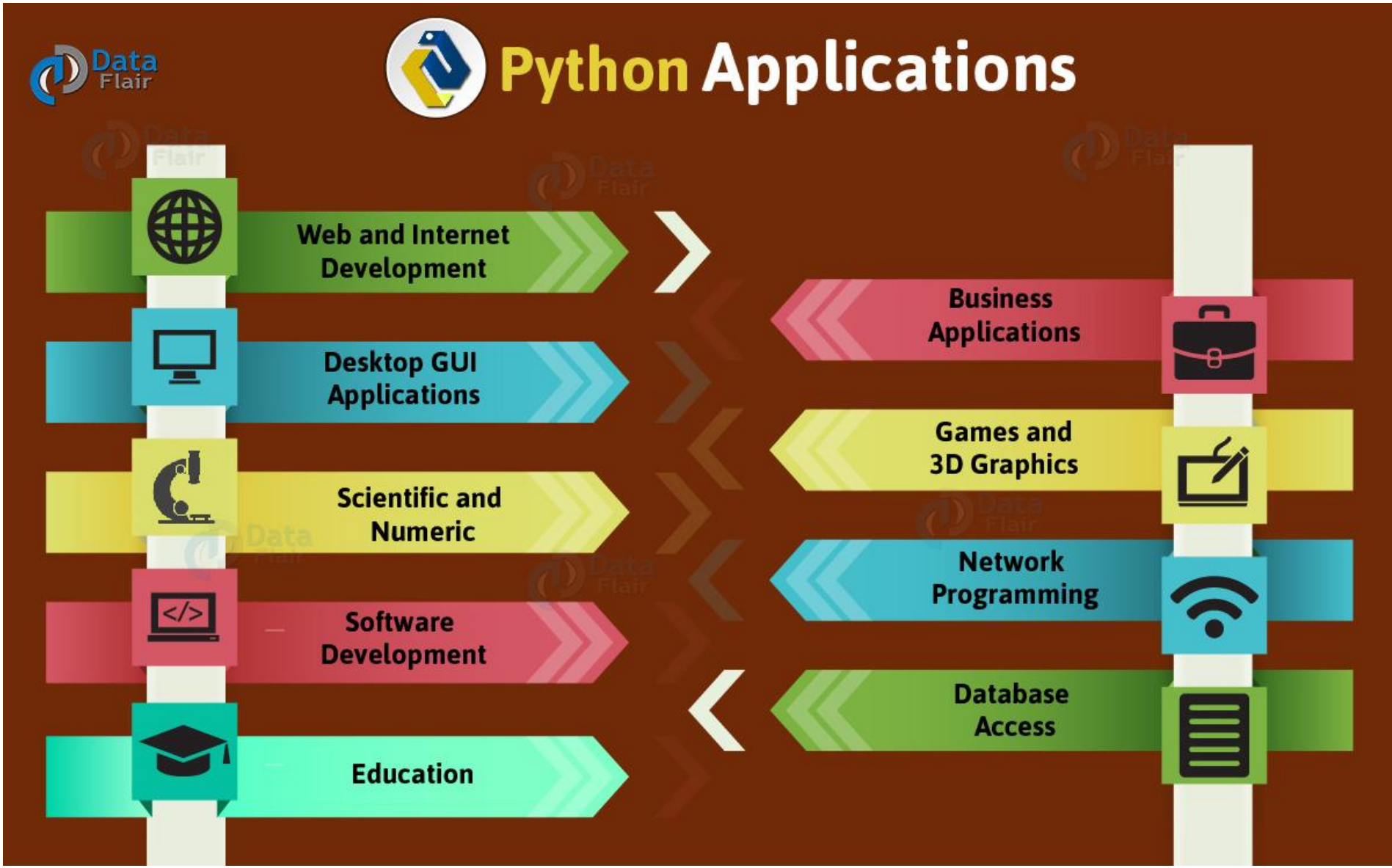
Hello World

- Open a terminal window and type “python”
- If on Windows open a Python IDE like IDLE
- At the prompt type ‘hello world!’

```
>>> 'hello world!'
'hello world!'
```

Python Features

- **Open source general-purpose language.**
- **Object Oriented, Procedural, Functional**
- **Easy to interface with C/ObjC/Java/Fortran**
- **Easy-ish to interface with C++ (via SWIG)**
- **Great interactive environment**



Source: https://data-flair.training/blogs/python-applications/

4 Major Versions of Python

- “Python” or “CPython” is written in C/C++
  - Version 2.7 came out in mid-2010
  - latest Version 3.7
  https://www.python.org/downloads/

- “Jython” is written in Java for the JVM
- “IronPython” is written in C# for the .Net environment

Go To Website

Development Environments

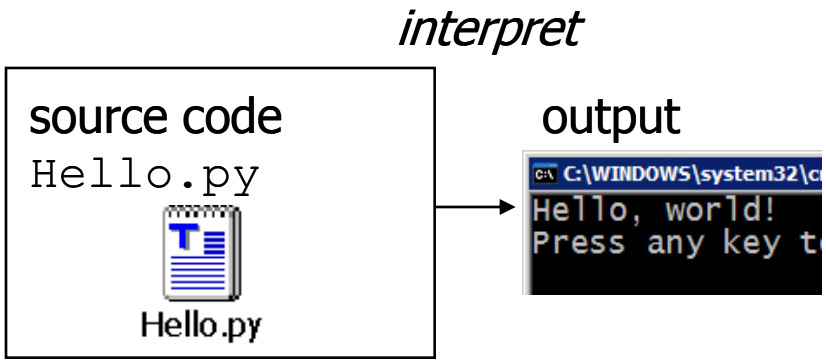
what IDE to use? http://stackoverflow.com/questions/81584

1. PyDev with Eclipse
2. Komodo
3. Emacs
4. Vim
5. TextMate
6. Gedit
7. Idle
8. PIDA (Linux)(VIM Based)
9. NotePad++ (Windows)
10. BlueFish (Linux)

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
>>>
```



PRINT

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

```
print ("Message")
print (Expression)
```

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

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

  - Prints several messages and/or expressions on the same line.
- Examples:

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

Output:

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

Documentation

The ‘#’ starts a line comment

```
>>> 'this will print'
'this will print'
>>> #'this will not'
>>>
```

Whitespace

Whitespace is meaningful in Python: especially indentation and placement of newlines

- Use a newline to end a line of code
  - Use `\` when must go to next line prematurely
- No braces `{}` to mark blocks of code, use *consistent* indentation instead
  - First line with *less* indentation is outside of the block
  - First line with *more* indentation starts a nested block
- Colons start of a new block in many constructs, e.g. function definitions, then clauses

No Braces

- Python uses ***indentation*** instead of braces to determine the scope of expressions
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope)
- This **forces** the programmer to use proper indentation since the indenting is part of the program!

Variables

- Are not declared, just assigned
- The variable is created the first time you assign it a value
- Are references to objects
- Type information is with the object, not the reference
- Everything in Python is an object

Multiple assignments

- In Python, multiple assignments can be made in a single statement as follows:
- a, b, c = 5, 3.2, "Hello"

Variables

- 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

```
>>> x = 7
>>> x
7
>>> x = 'hello'
>>> x
'hello'
>>>
```

- Data type is a property of the object and not of the variable

Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

```
bob   Bob   _bob   _2_bob_   bob_2   BoB
```

- There are some reserved words:

```
and, assert, break, class, continue,
def, del, elif, else, except, exec,
finally, for, from, global, if,
import, in, is, lambda, not, or,
pass, print, raise, return, try,
while
```

Naming conventions

The Python community has these recommend-ed naming conventions

- **joined_lower** for functions, methods and, attributes
- **joined_lower** or **ALL_CAPS** for constants
- **StudlyCaps** for classes
- **camelCase** only to conform to pre-existing conventions
- Attributes: interface, _internal, __private

Assignment

- You can assign to multiple names at the same time

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

This makes it easy to swap values

```
>>> x, y = y, x
```

- Assignments can be chained

```
>>> a = b = x = 2
```

EXPRESSIONS

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

  Examples: `1 + 4 * 3`

- 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`

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`

Basic Datatypes

- **Integers (default for numbers)**

```
z = 5 // 2  answer 2 integer division
    -5//2           -3
```

- **Floats**

```
x = 3.456
```

- **Strings**
  - Can use “” or ‘’ to specify with “abc” == ‘abc’
  - Unmatched can occur within the string: “matt’s”
  - **Use triple double-quotes** for multi-line strings or strings than contain both ‘ and “ inside of them: “““a‘b“c”””

What do you think happens when you try to do math with words?

- **>>> print 'red' + 'yellow'**
- **redyellow**

- **>>> print 'red' * 3**
- **redredred**
- **>>> print 'red' + str(3)**
- **red3**
- **>>> print 'red' + 3**
- **Traceback (most recent call last): File "", line 1, in TypeError: cannot concatenate 'str' and 'int' objects**
- **>>>**

Numbers: Integers

- Integer – the equivalent of a C long
- Long Integer – an unbounded integer value.

```
>>> 132224
132224
>>> 132323 ** 2
17509376329L
>>>
```

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
1.2323200000000001
>>> print 1.23232
1.23232
>>> 1.3E7
13000000.0
>>> int(2.0)
2
>>> float(2)
2.0
```

Numbers: Complex

- Built into Python
- Same operations are supported as integer and float

```
>>> x = 3 + 2j
>>> y = -1j
>>> x + y
(3+1j)
>>> x * y
(2-3j)
```

String Literals

- Strings are *immutable*
- There is no char type like in C++ or Java
- + is overloaded to do concatenation

```
>>> x = 'hello'
>>> x = x + ' there'
>>> x
'hello there'
```

STRINGS

- **string:** A sequence of text characters in a program.
  - Strings start and end with quotation mark " or apostrophe ' characters.
  - Examples:

```
"hello"
"This is a string"
"This, too, is a string.   It can be very long!"
```

- A string may not span across multiple lines or contain a " character.

```
"This is not
a legal String."
"This is not a "legal" String either."
```

- A string can represent characters by preceding them with a backslash.
  - \n = linefeed (prints the stuff after this on the next line)
    \r = carriage return (basically also used for printing stuff on the next line)
    \' = print a single quote (‘) in your text
    \" = print a double quote (“) in your text
    \\ = print a backslash (\ ) in your text
  - Example: print("Hello\tthere\nHow are you?")

```
In [54]: print('Hello\tthere\nHow are
you?')
Hello   there
How are you?
```

INDEXES

- Characters in a string are numbered with *indexes* starting at 0:
  - Example:

```
name = "P. Diddy"
```

index	0	1	2	3	4	5	6	7
character	P	.		D	i	d	d	y

- Accessing an individual character of a string:

variableName `[` ***index*** `]`

  - Example:

```
print (name, "starts with", name[0])
```

  Output:

```
P. Diddy starts with P
```

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

STRING PROPERTIES

- `len(`***string***`)` - number of characters in a string (including spaces)
- `str.lower(`***string***`)` - lowercase version of a string
- `str.upper(`***string***`)` - uppercase version of a string
- Example:

```
name = "Martin Douglas Stepp"
length = len(name)
big_name = str.upper(name)
print big_name, "has", length, "characters"
```

Output:

```
MARTIN DOUGLAS STEPP has 20 characters
```

Python List

- List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

```
a = [1, 2.2, 'python']
```

EXERCISE

- Write a program to perform following operation on two user input numbers.
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Modular division

Python Dictionary

- Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data.

```
d = {1:'value','key':2}
```

Python Set

- Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

```
a = {5,2,3,1,4}
a {1, 2, 3, 4, 5}
```

Python Tuple

- Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

```
t = (5,'program', 1+3j)        t[1]
                                'program'
```

String Literals: Many Kinds

- Can use single or double quotes, and three double quotes for a multi-line string

```
>>> 'I am a string'
'I am a string'
>>> "So am I!"
'So am I!'
>>> s = """And me too!
though I am much longer
than the others :)"""“
>>>print s
Output: 'And me too!\nthough I am much longer\nthan the others :)‘
>>> print s
And me too!
though I am much longer
than the others :)‘
```

Data Type Summary

- Integers: 2323, 3234L
- Floating Point: 32.3, 3.1E2
- Complex: 3 + 2j, 1j
- Lists: l = [1,2,3]
- Tuples: t = (1,2,3)
- Dictionaries: d = {‘hello’ : ‘there’, 2 : 15}

User Input

- The i**nput**(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods **int**(string), **float**(string), etc.

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
```

OUTPUT

print() function is used to output data to the standard output device (screen).

```
>>> print('This sentence is output to the screen')
This sentence is output to the screen
>>> a = 5
>>> print('The value of a is',a)
The value of a is 5
```

Output formatting:

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10
```

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

Output

```
>>> print('I love {0} and {1}'.format('bread','butter'))
I love bread and butter
>>> print('I love {1} and {0}'.format('bread','butter'))
I love butter and bread
```

Import

- When our program grows bigger, it is a good idea to break it into different modules. A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension .py.
- For example, we can import the math module by typing in import math.

```
>>> import math
>>> math.pi
3.141592653589793
```

Control Flow

Things that are False

- The boolean value False
- The numbers 0 (integer), 0.0 (float) and 0j (complex).
- The empty string "".
- The empty list [], empty dictionary {} and empty set set().

Things that are True

- The boolean value True
- All non-zero numbers.
- Any string containing at least one character.
- A non-empty data structure.

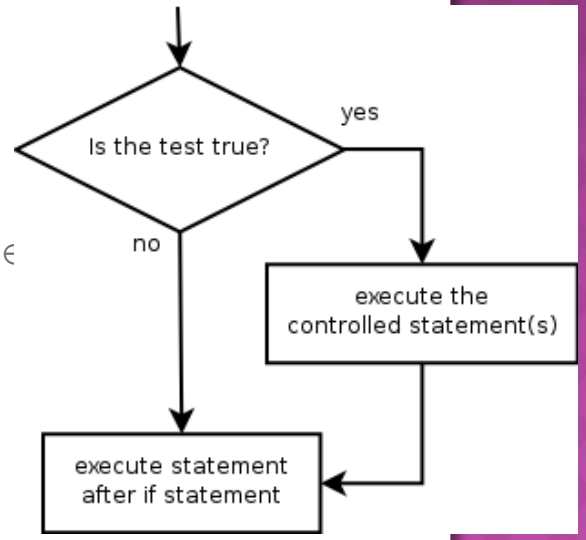
CONTROL FLOW: IF

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

```
if condition:
    statements
```

- Example:

```
gpa = 3.4
if gpa > 2.0:
    print ("Your application is accepte
```

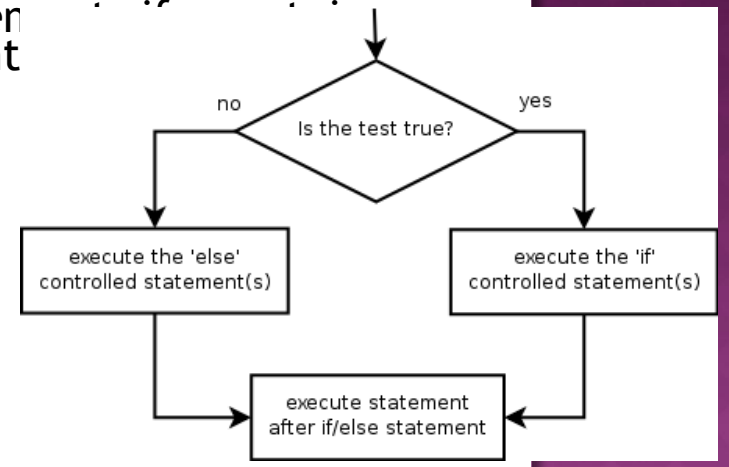


IF/ELSE

- **`if/else` statement**: Executes one block of stater condition is True, and a second block of statement

  - Syntax:

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

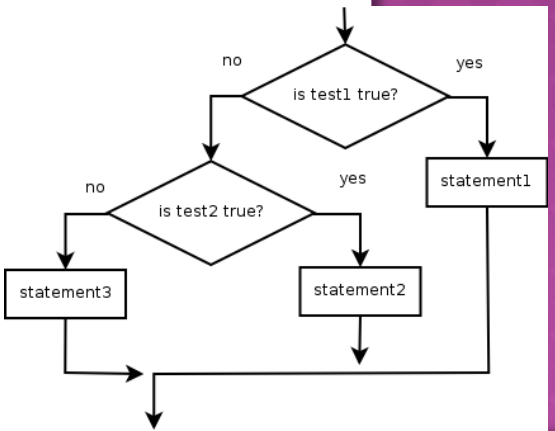


- Example:

```
gpa = 1.4
if gpa > 2.0:
    print ("Welcome to Mars University!")
else:
    print ("Your application is denied.")
```

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

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



```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message

num = float(input("Enter a number: "))
if num > 0:
     print("Positive number")
elif num == 0:
     print("Zero")
else:
     print("Negative number")
```

Output 1

```
Enter a number: 2
Positive number
```

Range Test

```
if (3 <= Time <= 5):
    print “Office Hour“
```

Or

```
if(Time>=3 and Time<=5)
```

A Code Sample (in IDLE)

```
x = 34 - 23                 # A comment.
y = “Hello”                 # Another one.
z = 3.45
if z == 3.45 or y == “Hello”:
    x = x + 1
    y = y + “ World”        # String concat.
print x
print y
```

Exercise

- **WAP to find if number entered by user is odd or even.**
- **WAP to check if the entered character is a vowel or a consonant.**
- **WAP to check if a year entered is leap year or not.**
- **Write a program that takes as input no. of the day in a week and outputs the corresponding name of the day. E.g. If input is day =2, output is "Day 2 of the week is Tuesday".**
- **Write a program to Input a four digit number and output it's each digit.**

WAP to find if number entered by user is odd or even.

```
no=int(input("enter the no"))
x=no//2
if x==0:
    print("even")
else:
    print("odd")
```

WAP to check if the entered character is a vowel or a consonant.

```
chr=input("enter character to check")
if chr=='a' or chr=='e' or chr=='i' or chr=='o' or chr=='u':
    print("vowel")
else:
    print("consonant")
```

Write a program to Input a four digit number and output it's each digit.

```
no=int(input("enter character to check"))
remain=no%10
quo=no//10
print("remain==",remain)
print("quo==",quo)
remain=quo%10
quo=quo//10
print("remain==",remain)
print("quo==",quo)
remain=quo%10
quo=quo//10
print("remain==",remain)
print("quo==",quo)
remain=quo%10
quo=quo//10
print("remain==",remain)
print("quo==",quo)
```

while

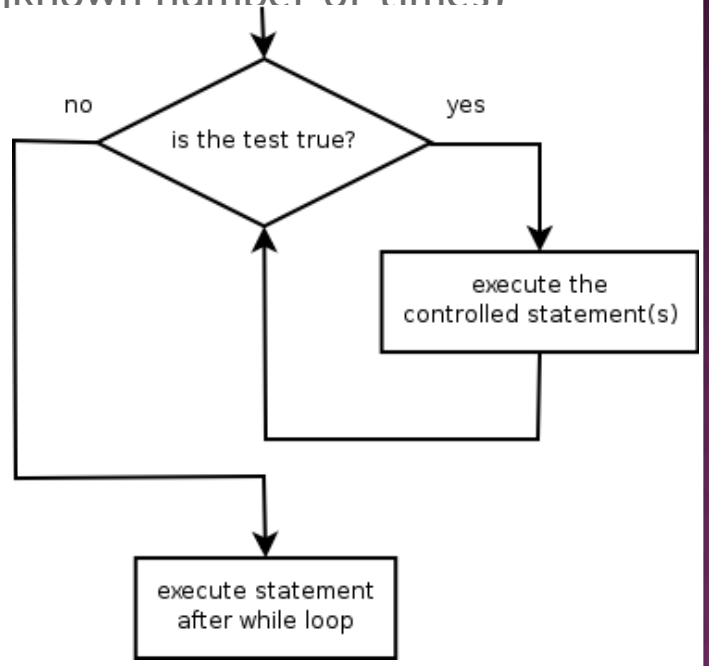
- **`while` loop**: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)

- Syntax:

```
while condition:
    statements
```

- Example:

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



  - Output:

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

While Loops

```
x = 1
while x < 10 :
    print x
    x = x + 1
```

In whileloop.py

```
>>> import whileloop
1
2
3
4
5
6
7
8
9
>>>
```

In interpreter

The Loop Else Clause

- The optional **else** clause runs only if the loop exits normally (not by break)

```
x = 1

while x < 3 :
    print x
    x = x + 1
else:
    print 'hello'
```

In whileelse.py

```
~: python whileelse.py
1
2
hello
```

Run from the command line

The Loop Else Clause

```
x = 1
while x < 5 :
    print x
    x = x + 1
    break
else :
    print 'i got here'
```

whileelse2.py

```
~: python whileelse2.py
1
```

```
# Program to add natural
# numbers upto n where
# n is provided by the user
# sum = 1+2+3+...+n

# take input from the user
n = int(input("Enter n: "))

# initialize sum and counter
sum = 0
i = 1

while i <= n:
     sum = sum + i
     i = i+1     # update counter

# print the sum
print("The sum is",sum)
```

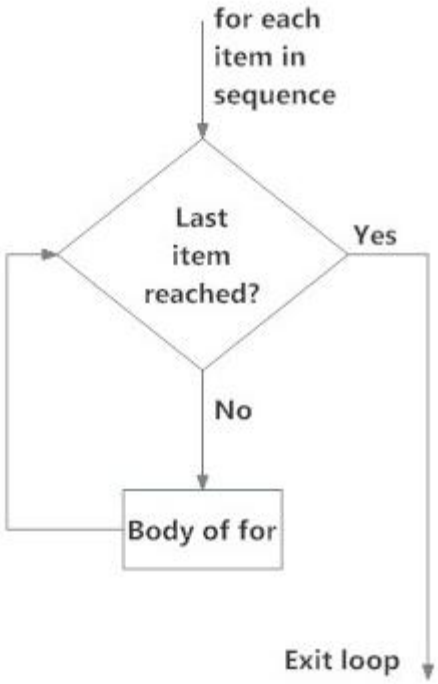
Output

```
Enter n: 10
The sum is 55
```

Syntax of for Loop

```
for val in sequence:
    Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.



```
# Program to find
# the sum of all numbers
# stored in a list

# List of numbers
numbers = [6,5,3,8,4,2,5,4,11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
     sum = sum+val

# print the sum
print("The sum is",sum)
```

Output

```
The sum is 48
```

```
# Program to show
# the control flow
# when using else block
# in a for loop

# a list of digit
list_of_digits = [0,1,2,3,4,5,6]

# take input from user
input_digit = int(input("Enter a digit: "))

# search the input digit in our list
for i in list_of_digits:
     if input_digit == i:
          print("Digit is in the list")
          break
else:
     print("Digit not found in list")
```

Output 1

break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. If it is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

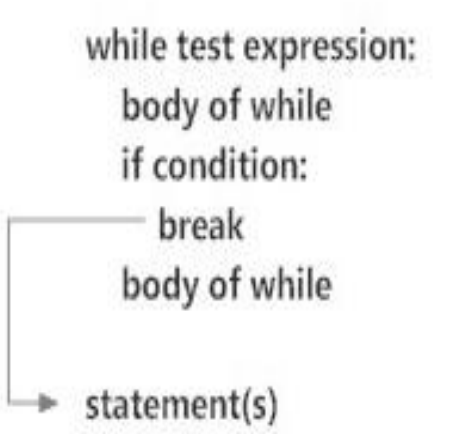


Fig: working of break in while loop

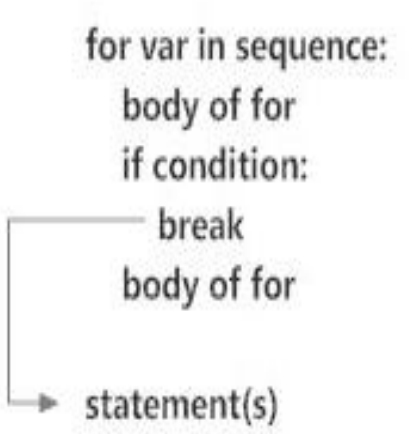
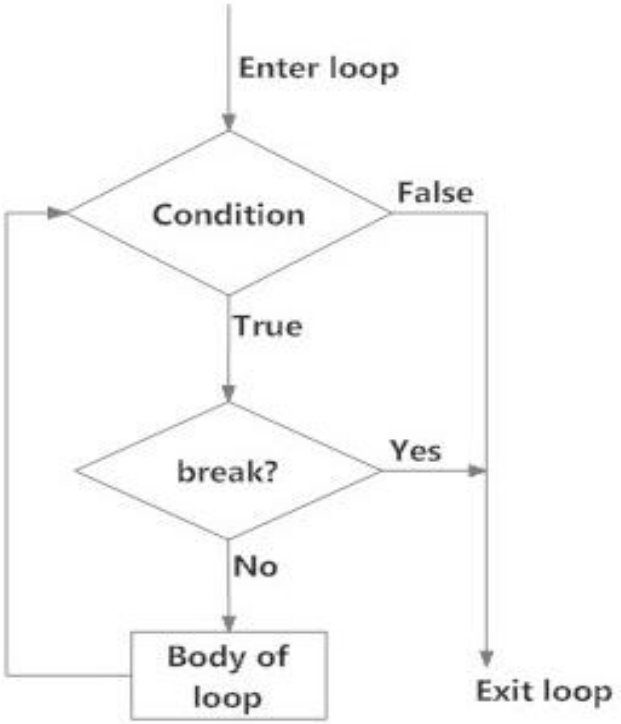


Fig: working of break in for loop



```
# Program to show the use of break statement inside loop

for val in "string":
     if val == "i":
          break
     print(val)

print("The end")
```

Output

```
 s
 t
 r
 The end
```

continue statement

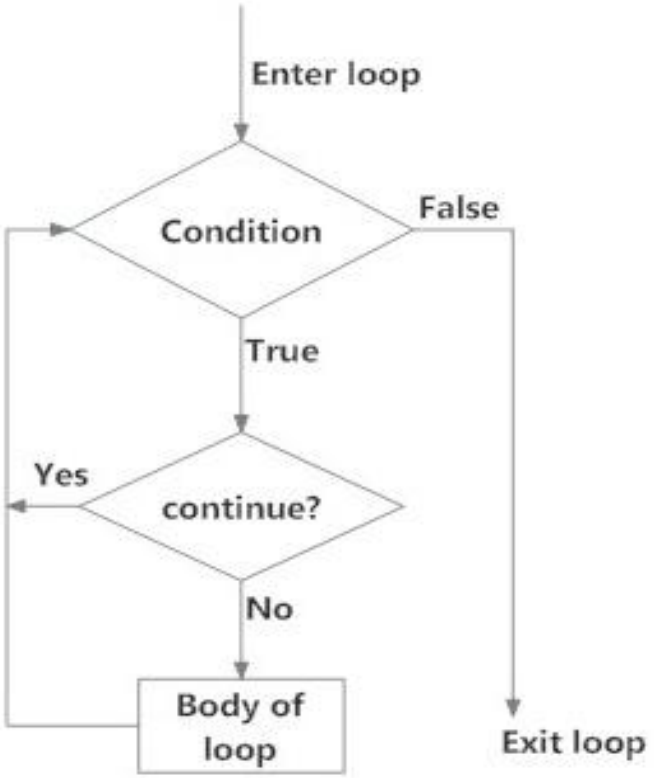
The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

```
while test expression:
    body of while
    if condition:
        continue
    body of while

statement(s)
```

```
for var in sequence:
    body of for
    if condition:
        continue
    body of for

statement(s)
```



```
# Program to show
# the use of continue
# statement inside loops

for val in "string":
     if val == "i":
          continue
     print(val)

print("The end")
```

Output

```
 s
 t
 r
 n
 g
 The end
```