

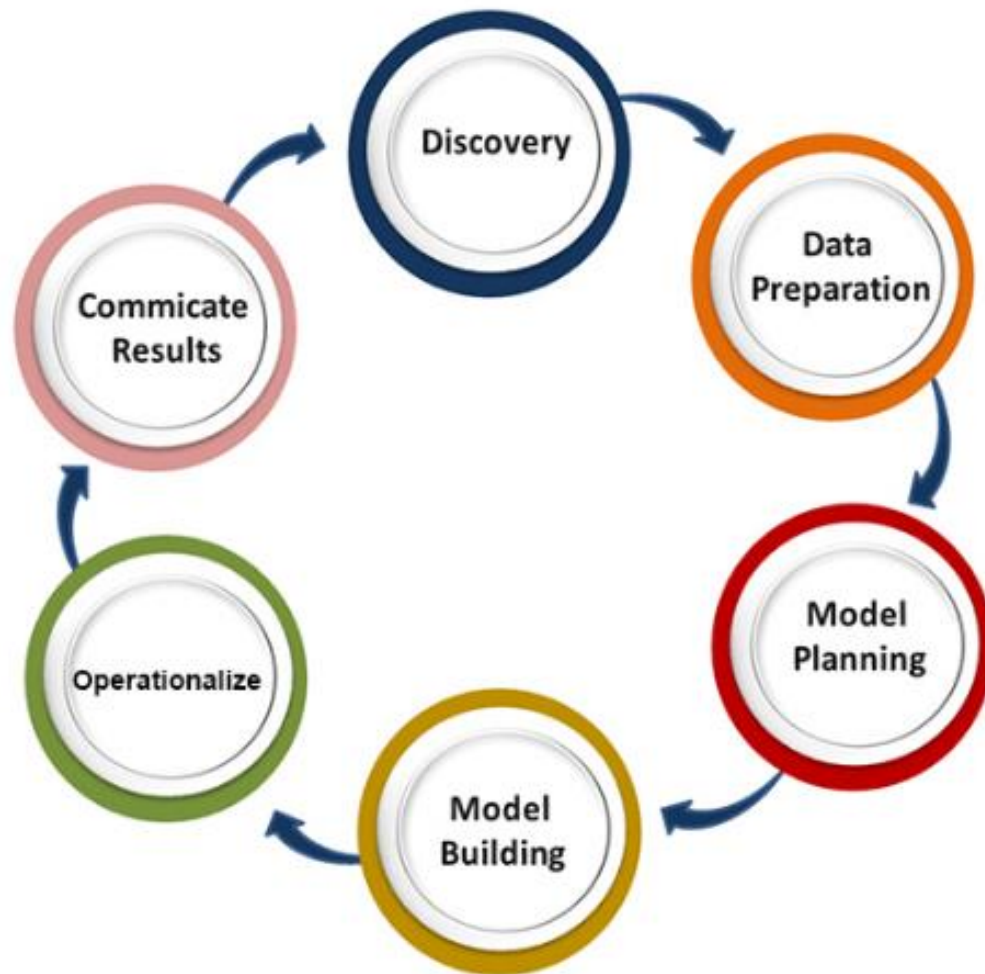
CSE3041-Programming for Data Science

Dr. Asnath Phamila Y
Professor,
School of Computer Science and Engineering,
VIT Chennai
asnathvicty.phamila@vit.ac.in

DATA SCIENCE

Data science is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from many structural and unstructured data

The life-cycle of data science is explained as below diagram.



Python – Why?

- Python has a simple syntax. Python programs are clear and easy to read. At the same time, Python provides powerful programming features, and is widely used.
- Companies and organizations that use Python include YouTube, Google, Yahoo, and NASA. Python is well supported and freely available at www.python.org.
- Guido van Rossum is the creator of the Python programming language, first released in the early 1990s. Its name comes from a 1970s British comedy sketch television show called *Monty Python's Flying Circus*.

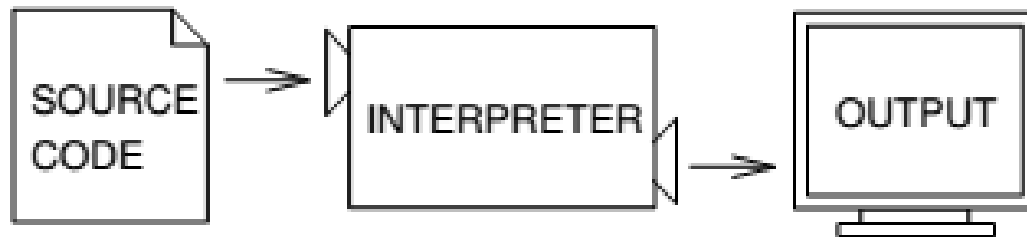


Python....

- Python is an example of a **high-level language**; other high-level languages you might have heard of are C, C++, Perl, and Java.
- There are also **low-level languages**, sometimes referred to as “machine languages” or “assembly languages.”
- Computers can only run programs written in low-level languages. So programs written in a high-level language have to be processed before they can run. This extra processing takes some time, which is a small disadvantage of high-level languages.

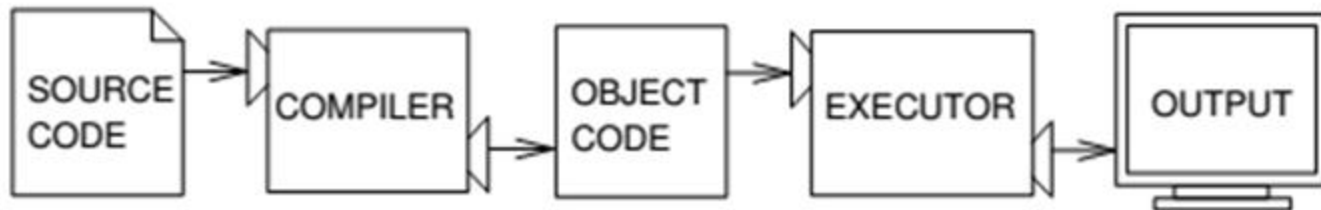
Python....

- Two kinds of program translator to convert from high-level languages into low-level languages: **interpreters** and **compilers**.
- An interpreter processes the program a little at a time, alternately reading lines and performing computations.



Python....

- A compiler reads the program and translates it completely before the program starts running. In this context, the high-level program is called the **source code**, and the translated program is called the **object code** or the **executable**. Once a program is compiled, you can execute it repeatedly without further translation.



A compiler translates source code into object code, which is run by a hardware executor.

Python....

- Python is considered an interpreted language because Python programs are executed by an interpreter. There are two ways to use the interpreter:
- **interactive mode** and **script mode**.

Python....Interactive mode

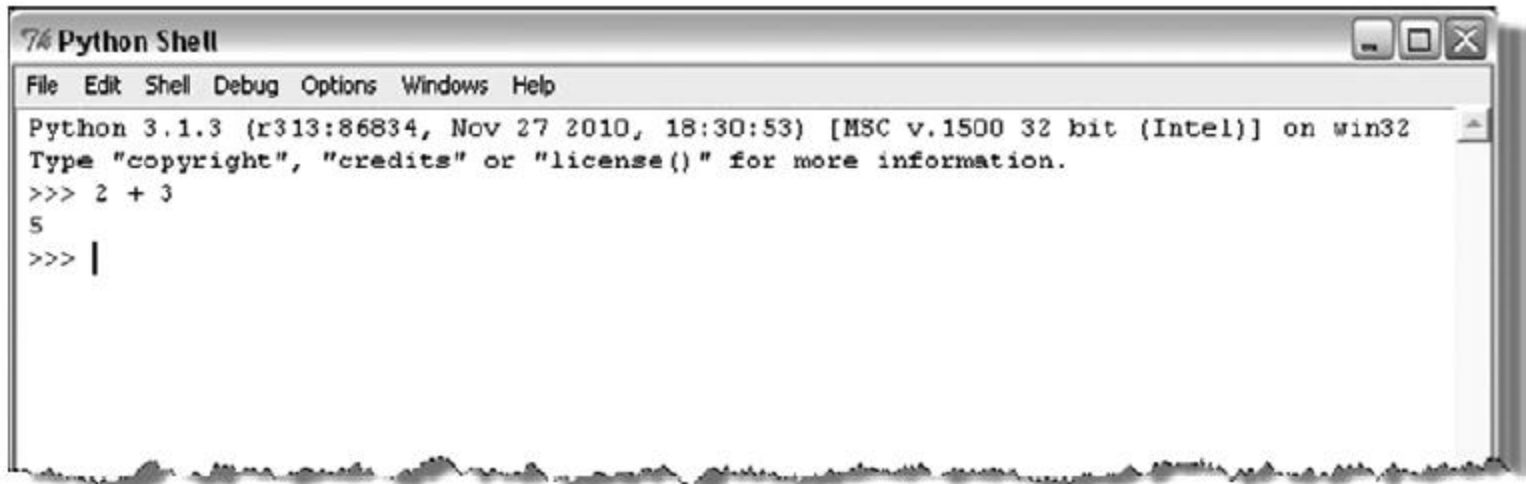
In interactive mode, you type Python programs and the interpreter displays the result:

```
>>> 1 + 1
```

```
2
```

The shell prompt, `>>>`, is the **prompt** the interpreter uses to indicate that it is ready. If you type `1 + 1`, the interpreter replies `2`.

Python Shell



The image shows a screenshot of a Windows-style application window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area displays the following content:

```
Python 3.1.3 (r313:86834, Nov 27 2010, 18:30:53) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 3
5
>>> |
```

The window has standard Windows controls (minimize, maximize, close) in the top right corner. The bottom of the window has a decorative, torn-paper-like edge.

Python.... Script Mode

- Alternatively, you can store code in a file and use the interpreter to execute the contents of the file, which is called a **script**.
- By convention, Python scripts have names that end with .py.
- Working in interactive mode is convenient for testing small pieces of code because you can type and execute them immediately.
- But for anything more than a few lines, you should save your code as a script so you can modify and execute it in the future.

Python....Script Mode ...

File name : first.py

```
print(4+3)
print(4-3)
print(4>3)
print("hello World")
```

```
C:\Python34\python.exe C:/Users/sathisbsk/first.py
```

```
7
```

```
1
```

```
True
```

```
hello World
```

```
Process finished with exit code 0
```

Keywords

- A **keyword** is an identifier that has pre-defined meaning in a programming language.
- Therefore, keywords cannot be used as “regular” identifiers. Doing so will result in a syntax error, as demonstrated in the attempted assignment to keyword and below,

```
>>> and = 10  
SyntaxError: invalid syntax
```

Keywords in Python

and	as	assert	break	class	continue	def
del	elif	else	except	finally	for	from
global	if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try	while
with	yield	false	none	true		

Variables and Identifiers

- A **variable** is a name (identifier) that is associated with a value.
- A simple description of a variable is “a name that is assigned to a value,”



Variables are assigned values by use of the **assignment operator**,

`num = 10`

`num = num + 1`

Rules for Identifier

- An **identifier** is a sequence of one or more characters used to provide a name for a given program element.
- Example : line, salary, emp1, emp_salary
- Python is *case sensitive*, thus, Line is different from line.
- Identifiers may contain letters and digits, but cannot begin with a digit.
- The underscore character, `_`, is also allowed to aid in the readability of long identifier names. It should not be used as the *first* character
- Spaces are not allowed as part of an identifier.

Identifier Naming

Valid Identifiers		Invalid Identifiers	Reason Invalid
totalSales		'totalSales'	quotes not allowed
totalsales		total sales	spaces not allowed
salesFor2010		2010Sales	cannot begin with a digit
sales_for_2010		_2010Sales	should not begin with an underscore

Comments

- Meant as documentation for anyone reading the code
- Single-line comments begin with the hash character ("#") and are terminated by the end of line.
- Python ignores all text that comes after # to end of line
- Comments spanning more than one line are achieved by inserting a multi-line string (with "'''" as the delimiter on each end) that is not used in assignment or otherwise evaluated, but sits in between other statements.
- *#This is also a comment in Python*
- "''" This is an example of a multiline comment that spans multiple lines ... "''"

```
a=int(input("Enter a"))
```

```
b=int(input("enter b"))
```

```
print(a,b)
```

```
Print(a+b)
```

Numeric literal

- A **numeric literal** is a literal containing only the digits 0–9, an optional sign character (1 or 2), and a possible decimal point. (The letter e is also used in exponential notation).
- If a numeric literal contains a decimal point, then it denotes a **floating-point value**, or “**float**” (e.g., 10.24); otherwise, it denotes an **integer value** (e.g., 10).
- *Commas are never used in numeric literals*

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> 1024
```

```
???
```

```
>>> -1024
```

```
???
```

```
>>> .1024
```

```
???
```

```
>>> 1,024
```

```
???
```

```
>>> 0.1024
```

```
???
```

```
>>> 1,024.46
```

```
???
```

String Literals

- **String literals**, or “**strings**,” represent a sequence of characters,

`'Hello' 'Smith, John' "Baltimore, Maryland 21210"`
- In Python, string literals may be *delimited* (surrounded) by a matching pair of either single (') or double (") quotes.
- ```
>>> print('Welcome to Python!')
```

```
>>>Welcome to Python!
```

# String Literal Values

`'A'`

- a string consisting of a single character

`'jsmith16@mycollege.edu'`

- a string containing non-letter characters

`"Jennifer Smith's Friend"`

- a string containing a single quote character

`' '`

- a string containing a single blank character

`''`

- the empty string

# Note

If this string were delimited with single quotes, the apostrophe (single quote) would be considered the matching closing quote of the opening quote, leaving the last final quote unmatched,

'Jennifer Smith's Friend' ... *matching quote*?

Thus, Python allows the use of more than one type of quote for such situations. (The convention used in the text will be to use single quotes for delimiting strings, and only use double quotes when needed.)



## LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> print('Hello')
???
```

```
>>> print('Hello")
???
```

```
>>> print('Let's Go')
???
```

```
>>> print("Hello")
???
```

```
>>> print("Let's Go!")
???
```

```
>>> print("Let's go!")
???
```

# Control Characters

- Special characters that are not displayed on the screen.
- *Control* the display of output
- Control characters do not have a corresponding keyboard character and represented by a combination of characters called an *escape sequence*.
- The backslash (\) serves as the escape character in Python.
- For example, the escape sequence '\n', represents the *newline control character*, used to begin a new screen line.

```
print('Hello\nJennifer Smith')
```

which is displayed as follows,

```
Hello
Jennifer Smith
```

## LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> print('Hello World')
???
```

```
>>> print('Hello World\n')
???
```

```
>>> print('Hello World\n\n')
???
```

```
>>> print('\nHello World')
???
```

```
>>> print('Hello\nWorld')
???
```

```
>>> print('Hello\n\nWorld')
???
```

```
>>> print(1, '\n', 2, '\n', 3)
???
```

```
>>> print('\n', 1, '\n', 2, '\n', 3)
???
```

# Data Types

- Python's data types are built in the core of the language
- They are easy to use and straightforward.
- Data types supported by Python
  - **Boolean values**
  - **Numbers**
  - **Strings**
  - **Tuples**
  - **Lists**
  - **Sets**
  - **Dictionaries**

# Basic Arithmetic operators in Python

---

| Command | Name                                 | Example | Output |
|---------|--------------------------------------|---------|--------|
| +       | Addition                             | 4 + 5   | 9      |
| -       | Subtraction                          | 8 - 5   | 3      |
| *       | Multiplication                       | 4 * 5   | 20     |
| /       | Division                             | 19 / 3  | 6.3333 |
| //      | Floor Division                       | 19//3   | 6      |
| %       | Remainder ( <a href="#">modulo</a> ) | 19 % 3  | 1      |
| **      | Exponent                             | 2 ** 4  | 16     |

# Order of Operations

| Operator | Operation      | Precedence |
|----------|----------------|------------|
| ()       | parentheses    | 0          |
| **       | exponentiation | 1          |
| *        | multiplication | 2          |
| /        | division       | 2          |
| //       | int division   | 2          |
| %        | remainder      | 2          |
| +        | addition       | 3          |
| -        | subtraction    | 3          |

# Conversion between different bases

- Provides built-in functions that allow you to convert integers to other bases' digit strings

```
>>> oct(64), hex(64), bin(64)
```

```
Numbers=>digit strings
```

- ('0o100', '0x40', '0b1000000')



# Input and output function

**Input function : input**

```
Basic_pay = input('Enter the Basic Pay: ')
```

**Output function : print**

```
>>> print('Hello world!')
```

```
print('Net Salary', salary)
```

# Input and output function

## Input function : input

```
Basic_pay = input('Enter the Basic Pay: ')
```

Input received **is always a string**.

To **convert it to integer** use int function of Python

```
Basic_pay = int(input('Enter the Basic Pay: '))
```

## Output function : print

```
>>> print('Hello world!')
```

```
print('Net Salary', salary)
```

# Type conversion...

```
line = input('How many credits do you have?')
num_credits = int(line)
line = input('What is your grade point average?')
gpa = float(line)
```

Here, the entered number of credits, say '24', is converted to the equivalent integer value, 24, before being assigned to variable `num_credits`. For input of the gpa, the entered value, say '3.2', is converted to the equivalent floating-point value, 3.2. Note that the program lines above could be combined as follows,

```
num_credits = int(input('How many credits do you have? '))
gpa = float(input('What is your grade point average? '))
```

# Repeated Print

```
>>>print('a'*15)
```

```
prints 'a' fifteen times
```

```
>>>print('\n'*15)
```

```
prints new line character fifteen times
```

# Different patterns in Algorithm

## Sequential

- **Sequential structure executes the program in the order in which they appear in the program**

## Selectional (conditional-branching)

- **Selection structure control the flow of statement execution based on some condition**

## Iterational (Loops)

- **Iterational structures are used when part of the program is to be executed several times**

# Problem 1

ABC company Ltd. is interested to computerize the pay calculation of their employee in the form of Basic Pay, Dearness Allowance (DA) and House Rent Allowance (HRA). DA is 80% of Basic Pay, and HRA is 30% of Basic pay. They have the deduction in the salary as PF which is 12% of Basic pay. Given the Basic pay, write a program to calculate the Gross pay.

# Problem 1

ABC company Ltd. is interested to computerize the pay calculation of their employee in the form of Basic Pay, Dearness Allowance (DA) and House Rent Allowance (HRA). DA is 80% of Basic Pay, and HRA is 30% of Basic pay. They have the deduction in the salary as PF which is 12% of Basic pay.

| Input     | Processing                                                                                              | Output | Solution Alternative |
|-----------|---------------------------------------------------------------------------------------------------------|--------|----------------------|
| Basic Pay | Calculate Salary<br>( Basic Pay + (<br>Basic Pay * 0.8) + (<br>Basic Pay * 0.3 - (<br>Basic Pay * 0.12) | Salary | Nothing              |

# Python code

#Enter the basic pay

```
bp=float (input('Enter the basic pay:'))
```

# net pay calucluation

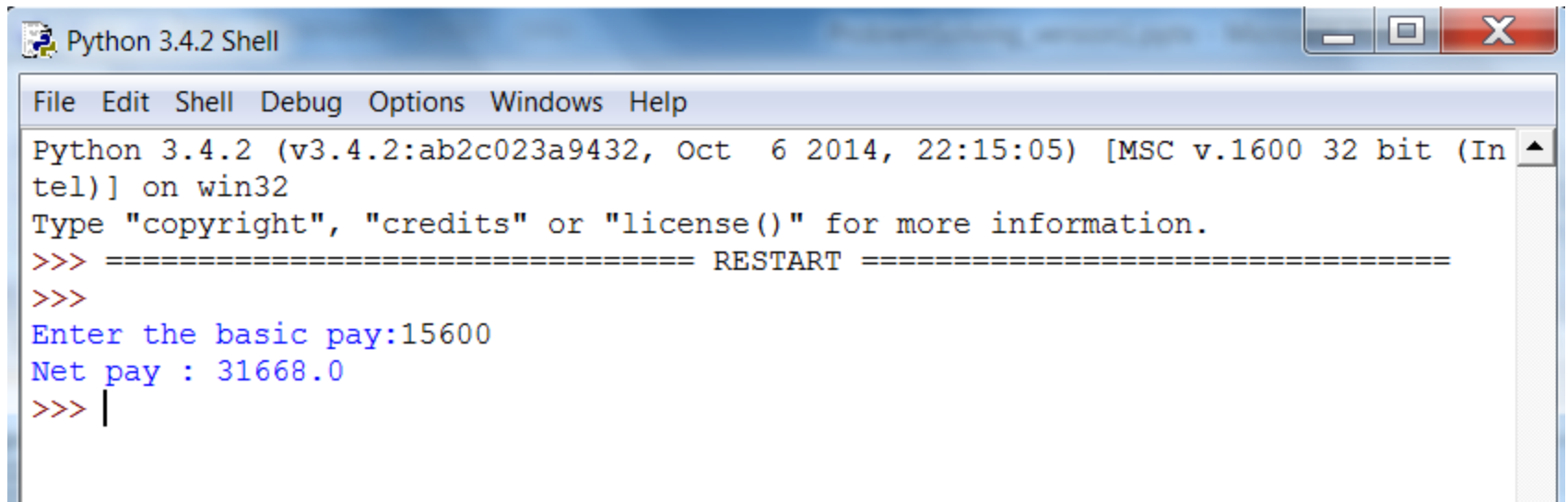
```
netpay =bp + (bp*0.8) + (bp*0.3) - (bp*0.12)
```

# display net salary

```
print ('Net pay :',netpay)
```



# Output



A screenshot of a Windows-style application window titled "Python 3.4.2 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area displays the following output:

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter the basic pay:15600
Net pay : 31668.0
>>> |
```

# Sequential Pattern

**Example1:** Find the average runs scored by a batsman in 4 matches

Algorithm:

**Step 1: Start**

**Step 2: Input 4 scores say `runs1,runs2,runs3` and `runs4`**

**Step 3: Accumulate `runs1,runs2,run3`,and `runs4` and store it  
in the variable called `total_runs`**

**Step 4: Divide `total_runs` by 4 and find the `average`**

**Step 5: Display the `average`**

**Step 6: Stop**

## Pseudo code:

Begin

read run1,run2,run3 and run4

compute  $\text{total\_run} = \text{run1} + \text{run2} + \text{run3} + \text{run4}$

compute  $\text{batting\_average} = \text{total\_run} / 4$

display batting\_average

end

# Batting Average

---

```
print("Enter four scores")
run1 = int(input())
run2 = int(input())
run3 = int(input())
run4 = int(input())
total_run=(run1+run2+run3+run4)
batting_average= total_run/4
print('batting_average is' ,batting_average)
```

# Area of a circle

Step 1 : Start

Step 2: Get the input for **RADIUS**

Step 3 : Find the square of **RADIUS** and store it in **SQUARE**

Step 4 : Multiply **SQUARE** with 3.14 and store the result in  
**AREA**

Step 5: Stop

## Pseudo code:

begin

accept radius

compute square = radius \* radius

compute area = pi \* square

display area

end

# Area of a circle

```
import math
print("Enter radius")
radius=float(input())
area = math.pi*radius*radius
print("area of circle is ", area)
```

## Exercise 2

My friend wants to make some curtains to windows in a home. Calculate how much material to buy for the given size of the window. Assume that the roll of material is going to be 140cm wide.



# Pseudocode

READ Length\_Of\_Window, Breadth\_Of\_Window,  
Number\_Of\_Window

COMPUTE Area\_Of\_Window = Length\_Of\_Window \*  
Breadth\_Of\_Window

COMPUTE Length\_Of\_Material = Area\_Of\_Window \*  
Number\_Of\_Window / 140

PRINT Length\_Of\_Material

# Python Program

```
#Read the input from the user
```

```
Length_Of_Window = float(input('Enter length of window in
cm'))
```

```
Breadth_Of_Window = float(input('Enter breadth of window in
cm'))
```

```
Number_Of_Window = int(input('Enter number of window'))
```

```
#calculate area of one window
```

```
Area_Of_Window = Length_Of_Window *
```

```
Breadth_Of_Window
```

# Python Program

```
#compute length of material required
Length_Of_Material = Area_Of_Window *
Number_Of_Window / 140
print('length of material required in cm',Length_Of_Material)
```

# Example Problem

- Little Bob loves chocolate, and he goes to a store with Rs.  $N$  in his pocket. The price of each chocolate is Rs.  $C$ . The store offers a discount: for every  $M$  wrappers he gives to the store, he gets one chocolate for free. This offer shall be availed only once. How many chocolates does Bob get to eat?

```
N=int(input())
```

```
C=int(input())
```

```
W=int(input())
```

```
NoC= N//C
```

```
Extra=NoC//W
```

```
Total_Chocolates = NoC + Extra
```

```
print(Total_Chocolates)
```

# Pseudocode

- READ Money\_In\_Hand and Cost\_Of\_Chocolate
- COMPUTE Number\_Of\_choc as  $\text{Money\_In\_Hand} / \text{Cost\_Of\_Chocolate}$
- $\text{Number\_Of\_choc} = \text{Number\_Of\_choc} + (\text{Number\_Of\_choc} / \text{Wrapper\_Offer})$
- PRINT num\_of\_chocolates

## Exercise 4

- Write a program to compute the difference between any two digit number and its reverse. For example, for the number 54, its reverse is 45 and the difference between them is 9.

# Python Program

#Read the number

```
Number = int(input('Enter Number'))
```

# number mod 10 gives the second digit and floor division (//)

gives first digit

```
Reverse = 10*(Number%10) + Number//10
```

#abs is built in function in Python

```
Difference = abs(Number-Reverse)
```

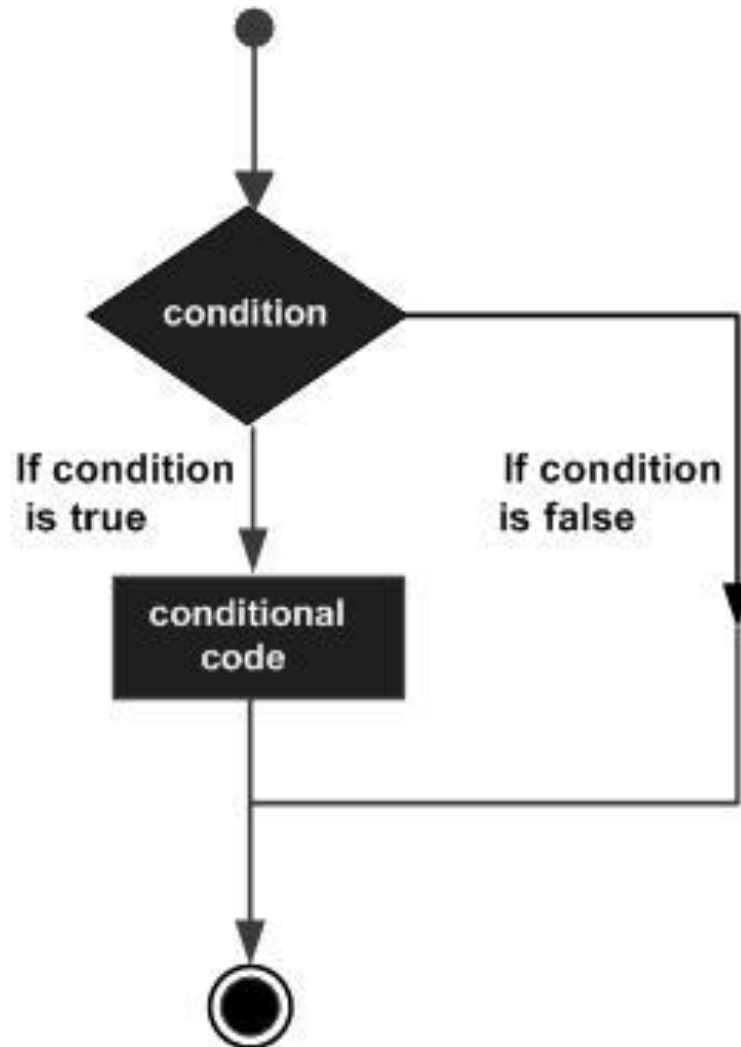
```
print(Difference)
```



# Selection pattern

- A **selection control statement** is a control statement providing selective execution of instructions.

# Control flow of decision making



# If Statement

- An **if statement** is a selection control statement based on the value of a given Boolean expression.

The if statement in Python is

| If statement                                       | Example use                                               |
|----------------------------------------------------|-----------------------------------------------------------|
| If condition:<br>statements<br>else:<br>statements | If grade >=50:<br>print('pass')<br>else:<br>print('fail') |

# If Statement

| If statement                                                             | Example use                                                                        |
|--------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| <pre>If condition:<br/>    statements<br/>else:<br/>    statements</pre> | <pre>If grade &gt;=50:<br/>    print('pass')<br/>else:<br/>    Print('fail')</pre> |

# Indentation in Python

- One fairly unique aspect of Python is that the amount of indentation of each program line is significant.
- In Python indentation is used to associate and group statements

## Valid indentation

(a) 

```
if condition:
 statement
 statement
else:
 statement
 statement
```

(b) 

```
if condition:
 statement
 statement
else:
 statement
 statement
```

## Invalid indentation

(c) 

```
if condition:
 statement
 statement
else:
 statement
 statement
```

(d) 

```
if condition:
 statement
 statement
else:
 statement
 statement
```

# Nested if Statements

- There are often times when selection among more than two sets of statements (suites) is needed.
- For such situations, if statements can be nested, resulting in **multi-way selection**.

| Nested if statements                                                                                                                                   | Example use                                                                                                                                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>if condition:     statements else:     if condition:         statements     else:         if condition:             statements         etc.</pre> | <pre>if grade &gt;= 90:     print('Grade of A') else:     if grade &gt;= 80:         print('Grade of B')     else:         if grade &gt;= 70:             print('Grade of C')         else:             if grade &gt;= 60:                 print('Grade of D')             else:                 print('Grade of F')</pre> |

# Else if Ladder

```
if grade >= 90:
 print('Grade of A')
elif grade >= 80:
 print('Grade of B')
elif grade >= 70:
 print('Grade of C')
elif grade >= 60:
 print('Grade of D')
else:
 print('Grade of F')
```

# Multiple Conditions

- Multiple conditions can be checked in an 'if' statement using logical operators 'and' and 'or'.
- Python code to print 'excellent' if mark1 and mark2 is greater than or equal to 90, print 'good' if mark1 or mark2 is greater than or equal to 90, print 'need to improve' if both mark1 and mark2 are lesser than 90



Write a Python code to  
print 'excellent' if mark1 and mark2 are greater  
than or equal to 90,  
print 'good' if mark1 or mark2 is greater than or  
equal to 90,  
print 'need to improve' if both mark1 and mark2  
are lesser than 90

```
if mark1>=90 and mark2 >= 90:
 print('excellent')
elif mark1>=90 or mark2 >= 90:
 print('good')
else:
 print('need to improve')
```

# Ternary Operator

## Syntax

x if y else z

Eg:

```
>>> a = 2
```

```
>>> b = 1 if a > 1 else 0
```

```
>>> b
```

```
1
```

```
>>> a = 1
```

```
>>> b = 1 if a > 1 else 0
```

```
>>> b
```

```
0
```

a=10

b=20

maxnum=a if (a>b) else b

print(maxnum)

# Greatest of three numbers

a=30

b=20

c=35

```
maxnum=(a if(a>c) else c) if (a>b) else (b if(b>c) else c)
print(maxnum)
```

# Browsing Problem

Given the number of hours and minutes browsed, write a program to calculate bill for Internet Browsing in a browsing center. The conditions are given below.

(a) 1 Hour Rs.50

(b) 1 minute Re. 1

(c) Rs. 200 for five hours

**Boundary condition:** User can only browse for a maximum of 7 hours

Check boundary conditions

# Pseudocode

READ hours and minutes

SET amount = 0

IF hours  $\geq$  5 then

    CALCULATE amount as amount + 200

    COMPUTE hours as hours – 5

END IF

COMPUTE amount as amount + hours \* 50

COMPUTE amount as amount + minutes \* 1

PRINT amount

```
hours=int(input("Enter Hours"))
mins=int(input("Enter Minutes"))
if(hours>7):
 print("Invalid Input")
elif hours>=5:
 amount=200+(hours-5)*50+mins
 print(amount)
else:
 amount=hours*50+mins
 print(amount)
```



# Built-in format Function

- Because floating-point values may contain an arbitrary number of decimal places, the built-in **format** function can be used to produce a numeric string version of the value containing a specific number of decimal places.

```
>>> 12/5
```

```
2.4
```

```
>>> format(12/5, '.2f')
```

```
'2.40'
```

```
>>> 5/7
```

```
0.7142857142857143
```

```
>>> format(5/7, '.2f')
```

```
'0.71'
```

- In these examples, *format specifier* `'.2f'` rounds the result to two decimal places of accuracy in the string produced.

- For very large (or very small) values 'e' can be used as a format specifier,

```
>>> format(2 ** 100, '.6e')
'1.267651e+30'
```

## LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> format(11/12, '.2f')
```

```
???
```

```
>>> format(11/12, '.2e')
```

```
???
```

```
>>> format(11/12, '.3f')
```

```
???
```

```
>>> format(11/12, '.3e')
```

```
???
```

# Python is a Dynamic Type language

- In Python the same variable can be associated with values of different type during program execution, as indicated below.
- It's also very dynamic as it rarely uses what it knows to limit variable usage

|                            |         |
|----------------------------|---------|
| <code>var = 12</code>      | integer |
| <code>var = 12.45</code>   | float   |
| <code>var = 'Hello'</code> | string  |

## LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> num = 10
```

```
>>> num
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> num = 20
```

```
>>> num
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> k = num
```

```
>>> k
```

```
???
```

```
>>> id(k)
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> k = 30
```

```
>>> k
```

```
???
```

```
>>> num
```

```
???
```

```
>>> id(k)
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> k = k + 1
```

```
>>> k
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> id(k)
```

```
???
```

- Python also provides both **built-in functions and standard library modules** for numeric processing.
- The `pow` and `abs` built-in functions, for instance, compute powers and absolute values, respectively.
- Examples of the built-in `math` module

```
>>> import math
```

```
>>> math.pi, math.e # Common constants
(3.141592653589793, 2.718281828459045)
```

- `>>> math.sin(2 * math.pi / 180)`

`# Sine, tangent, cosine`

`0.03489949670250097`

`>>> math.sqrt(144), math.sqrt(2)`

`# Square root`

`(12.0, 1.4142135623730951)`

```
import math
radius = int(input('EnterRadius'))
area = math.pi*radius*radius
print("area is",area)
print(format(area,'.2f'))
```



# Example Problem

- Write a Python program to check whether a blood donor is eligible or not for donating blood. The conditions laid down are as under. Use if statement.
  - (a) Age should be above 18 years but not more than 55 years.
  - (b) Weight should be more than 45 kg.

# Python Program

```
age = int(input('Enter age'))
weight = float(input('Enter weight'))
donate = 'donate' if (age > 18 and age <=55) and
 (weight>45) else 'do not donate'
print(donate)
```

*Table 11-2. Augmented assignment statements*

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| $X += Y$  | $X \&= Y$ | $X -= Y$  | $X  = Y$  |
| $X *= Y$  | $X ^= Y$  | $X /= Y$  | $X >>= Y$ |
| $X \%= Y$ | $X <<= Y$ | $X **= Y$ | $X //= Y$ |

---

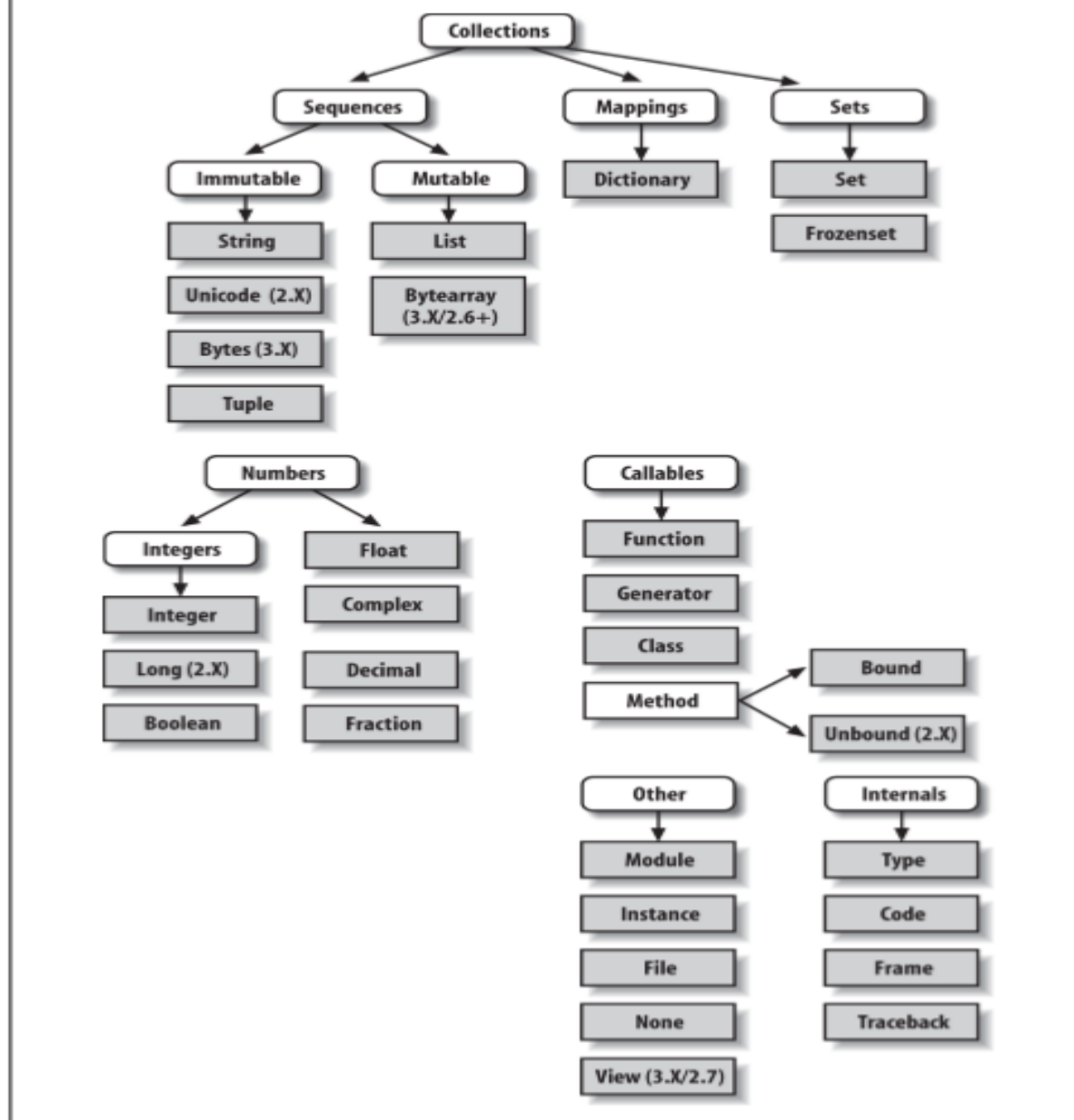


Figure 9-3. Python's major built-in object types, organized by categories. Everything is a type of object in Python, even the type of an object! Some extension types, such as named tuples, might belong in this figure too, but the criteria for inclusion in the core types set are not formal.

# Operators in Python

| Operators                                             | Description                                          |
|-------------------------------------------------------|------------------------------------------------------|
| <code>yield x</code>                                  | Generator function send protocol                     |
| <code>lambda args: expression</code>                  | Anonymous function generation                        |
| <code>x if y else z</code>                            | Ternary selection (x is evaluated only if y is true) |
| <code>x or y</code>                                   | Logical OR (y is evaluated only if x is false)       |
| <code>x and y</code>                                  | Logical AND (y is evaluated only if x is true)       |
| <code>not x</code>                                    | Logical negation                                     |
| <code>x in y, x not in y</code>                       | Membership (iterables, sets)                         |
| <code>x is y, x is not y</code>                       | Object identity tests                                |
| <code>x &lt; y, x &lt;= y, x &gt; y, x &gt;= y</code> | Magnitude comparison, set subset and superset;       |
| <code>x == y, x != y</code>                           | Value equality operators                             |

# Operators in Python

`x | y`

Bitwise OR, set union

`x ^ y`

Bitwise XOR, set symmetric difference

`x & y`

Bitwise AND, set intersection

`x << y, x >> y`

Shift x left or right by y bits

`x + y`

Addition, concatenation;

`x - y`

Subtraction, set difference

`x * y`

Multiplication, repetition;

`x % y`

Remainder, format;

`x / y, x // y`

Division: true and floor

`-x, +x`

Negation, identity

`~x`

Bitwise NOT (inversion)

# Operators in Python

|                       |                                                    |
|-----------------------|----------------------------------------------------|
| <code>x ** y</code>   | Power (exponentiation)                             |
| <code>x[i]</code>     | Indexing (sequence, mapping, others)               |
| <code>x[i:j:k]</code> | Slicing                                            |
| <code>x(...)</code>   | Call (function, method, class, other callable)     |
| <code>x.attr</code>   | Attribute reference                                |
| <code>(...)</code>    | Tuple, expression, generator expression            |
| <code>[...]</code>    | List, list comprehension                           |
| <code>{...}</code>    | Dictionary, set, set and dictionary comprehensions |

---

# Exercise 1

A General Service company hired you to calculate the labour cost and total charge for the services rendered to their client based on the formula given below:

Labour cost : Rate per hour \* hours worked

Total Charge : Labour cost + Cost of Materials.

Write a program to implement the above.



# Python Program

#get the details from user

Rate\_Per\_Hour = float(input('Enter Rate per hour'))

Number\_Of\_Hours\_Worked = float(input('Enter Number of  
hour Worked'))

Cost\_Of\_Material = float(input('Cost of material'))

#calculate the total charges using formulae

Total\_Charges = Rate\_Per\_Hour \*

Number\_Of\_Hours\_Worked + Cost\_Of\_Material

#print total charges

print (Total\_Charges)

# Exercise 3

- A professor is conducting a course on Discrete Mathematics to a class of 5 students. He is angry at the lack of their discipline, and he decides to cancel the class if there are fewer than 3 students present after the class starts.
- Given the arrival time of '5' students, find out if the class gets cancelled or not. Positive value of 'p' as input indicate that the student has come earlier by 'p' minutes to the class and negative value of 'p' as input indicate that the student has come late by 'p' minutes.

# Pseudocode

READ Arrival\_Time1

LET Count = 0

If Arrival\_Time1 < 0 THEN

    Count = Count + 1

READ Arrival\_Time2

If Arrival\_Time2 < 0 THEN

    Count = Count + 1

READ Arrival\_Time3

If Arrival\_Time3 < 0 THEN

    Count = Count + 1

# Pseudocode

READ Arrival\_Time4

If Arrival\_Time4 < 0 THEN

    Count = Count + 1

READ Arrival\_Time5

If Arrival\_Time5 < 0 THEN

    Count = Count + 1

IF Count > 2 THEN

    PRINT 'Class Cancelled'

ELSE

    PRINT 'Class Not Cancelled'

# Python Program

#get arrival time of students and increment count for late comers

# if value entered is less than zero then they are late comers

Arrival\_Time1 = int(input('Enter arrival time of first student'))

count = 1 if Arrival\_Time1<0 else 0

Arrival\_Time2 = int(input('Enter arrival time of second student'))

count = count+1 if Arrival\_Time2<0 else count

Arrival\_Time3 = int(input('Enter arrival time of third student'))

count = count+1 if Arrival\_Time3<0 else count

# Python Program

```
Arrival_Time4 = int(input('Enter arrival time of fourth student'))
```

```
count = count+1 if Arrival_Time4<0 else count
```

```
Arrival_Time5 = int(input('Enter arrival time of fifth student'))
```

```
count = count+1 if Arrival_Time5<0 else count
```

```
#when number of late comers is more than two class is
cancelled
```

```
Class_Status = 'cancelled' if count>2 else 'not cancelled'
```

```
print (Class_Status)
```

# Exercise Problem

1. Write a python code to check whether a given number is odd or even?
2. Write a python code to check whether a given year is leap year or not?
3. Write a python code in finding the roots of a quadratic equation?
4. Write a python program to segregate student based on their CGPA. The details are as follows:

$\leq 9$  CGPA  $\leq 10$  - outstanding

$\leq 8$  CGPA  $< 9$  - excellent

$\leq 7$  CGPA  $< 8$  - good

$\leq 6$  CGPA  $< 7$  - average

$\leq 5$  CGPA  $< 6$  - better

CGPA  $< 5$  - poor