Programming for Data Science (CSE3041)

Ramesh Ragala

VIT Chennai Campus

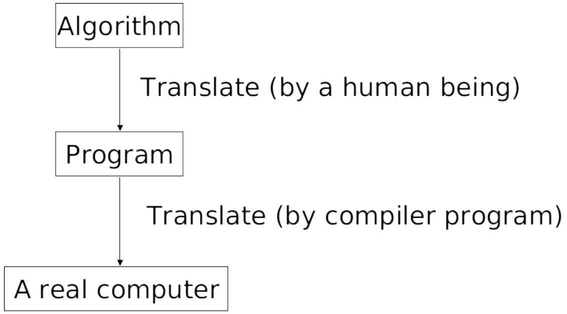
July 16, 2020

Need Of Programming Language

- Computers can execute tasks very rapidly.
- They can handle a greater amount of input data than human being.
- But they can not design a strategy to solve problems for you.
- Need one programming language to communicate with computer to solve the problem.
- What is a Programming?
  - Usually, one or more algorithms written in a programming language that can be translated to run on a real machine. $\Rightarrow$ sometimes called as software.
  - A programming language is somewhat like a natural language, but with a very limited set of statements and strict syntax rules.
  - Has statements to implement sequential, conditional and iterative processing algorithms.

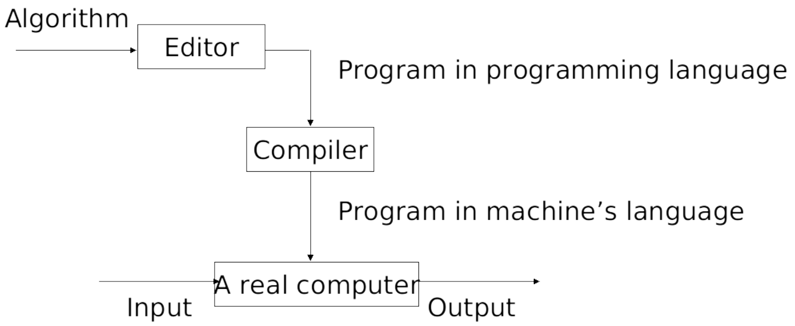
Need of Programming Language

- Algorithm to Hardware



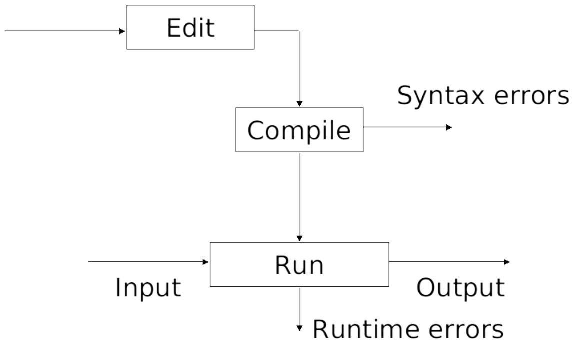
Need of Programming Language

- Program Development Process (**Data Flow**)


Algorithm
Editor
Program in programming language
Compiler
Program in machine's language
A real computer
Input
Output

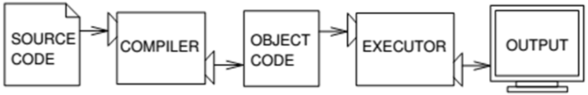
Need of Programming Language

- Program Development Process (**Control Flow**)



Compiler

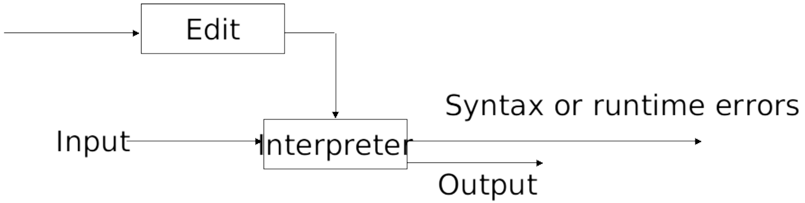
- It is a program that converts a program written in a programming language into a program in the native language, called machine language, of the machine that is to execute the program.
- It reads the program and translates it completely before the program starts running.
- High Level Program is called Source Code
- Translated Program is called Object Code or Executable Code



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

Interpreter

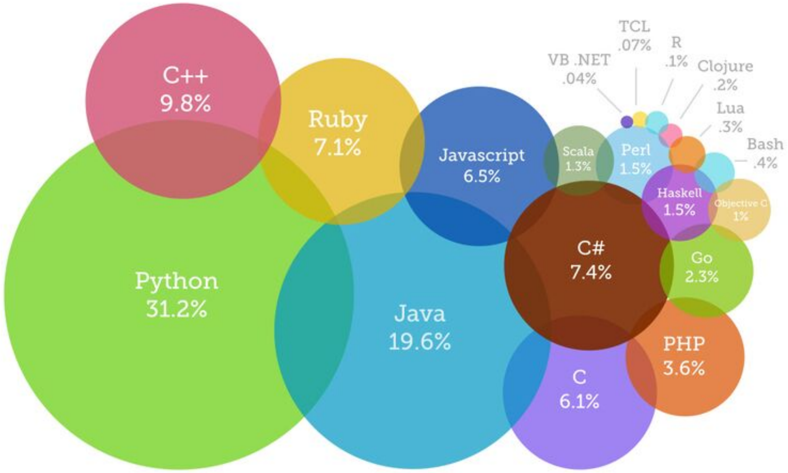
- Interpreter
  - It takes our program's statements at a time (one by one) and executes a corresponding set of machine instructions.
  - It is alternative to compiler.
  - The processing is slow.



Types of Error in Program Languages

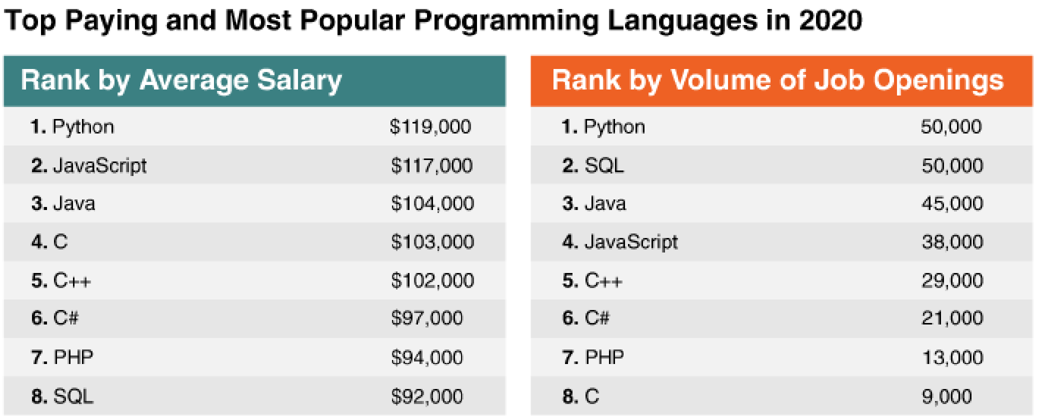
- Syntax Errors:
  - Statement does not obey the rules of programming language.
  - It refers to the structure of a program and the rules about that structure.
  - Some statement in the program is not a legal statement in the language.
  - Example: INT c; declaration in C language
- Run Time Errors
  - These error does not appear until after the program has started running.
  - These errors are also called exceptions.
  - An error occurs while the program is executing, causing the program to terminate (divide by zero, etc.)
  - Due to these error, program will terminate abnormally.
- Logical Errors/Semantic Errors:
  - The program executes to completion, but gives incorrect results.
  - We need to change the logic of the program to get correct results.

- Python Ranking



Python for Problem Solving

- Python Ranking



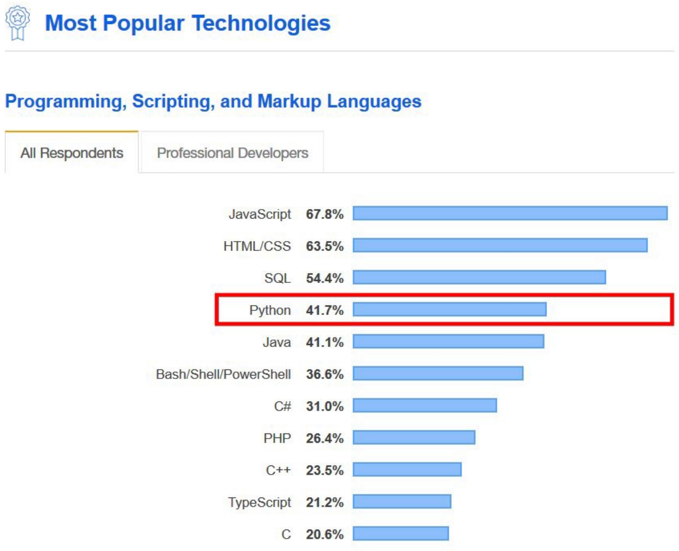
Top Paying and Most Popular Programming Languages in 2020

Rank by Average Salary	
1. Python	$119,000
2. JavaScript	$117,000
3. Java	$104,000
4. C	$103,000
5. C++	$102,000
6. C#	$97,000
7. PHP	$94,000
8. SQL	$92,000

Rank by Volume of Job Openings	
1. Python	50,000
2. SQL	50,000
3. Java	45,000
4. JavaScript	38,000
5. C++	29,000
6. C#	21,000
7. PHP	13,000
8. C	9,000

Python for Problem Solving

- Python Ranking according to IEEE



Python Users

Google

YouTube

NASA

YAHOO!
Maps

Walt Disney
FEATURE ANIMATION
ANIMATION TRAINING PROGRAMS

redhat

NOKIA
Connecting People

Python for Problem Solving

- Python has a **simple syntax** and very **few keywords**.
- Python programs are clear and easy to **read** and **Understand**.
- It has **Powerful programming features** and **highly portable** and **extensible**
- Python is a **High Level Language**.
- Machine Languages or Assembly Languages are referred as Low Level Languages
- High Level Languages have to be processed before they can run. $\rightarrow$ extra time.
- Two types of programs translators to convert High Level Program to Low Level program
    - Compiler
    - Interpreter

Brief History of Python

- Invented in the Netherlands, early 90s by **Guido van Rossum**.
- Named after Monty Python.
- Open sourced from the beginning.
- Considered a scripting language, but is much more Scalable, object oriented and functional from the beginning.
- It is a object oriented programming language. $\rightarrow$ everything is object.
- Genealogy:
  - Setl (NYU, J.Schwartz et al. 1969-1980).
  - ABC (Amsterdam, Meertens et al. 1980-).
  - Python (Van Rossum et all. 1996-).

Time Line of Python

- Python born, name picked - Dec 1989.
- First public release (USENET) - Feb 1991
- python.org website - 1996 or 1997
- 2.0 released - 2000
- Python Software Foundation - 2001
- ...
- ...
- 2.4 released - 2004
- 2.5 released – 2006
- Current version: 3.5.2 and 2.7.12

Introduction to Python

- Object oriented language

Introduction to Python

- Object oriented language
- Interpreted language

Introduction to Python

- Object oriented language
- Interpreted language
- Supports dynamic data type

Introduction to Python

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms

Introduction to Python

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time

Introduction to Python

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar

Introduction to Python

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- It's free Invalid Identifiers(open source)

Introduction to Python

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- It's free Invalid Identifiers(open source)
- Automatic memory management

Introduction to Python

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- It's free Invalid Identifiers(open source)
- Automatic memory management
- Glue language Interactive front-end for FORTRAN/C/C++ code

Python Language Properties

- Everything is an object

Python Language Properties

- Everything is an object
- Modules, classes, functions

Python Language Properties

- Everything is an object
- Modules, classes, functions
- Exception handling

Python Language Properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism

Python Language Properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping

Python Language Properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading

Python Language Properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure

Where to use Python

- System management (i.e., scripting)

Where to use Python

- System management (i.e., scripting)
- Graphic User Interface (GUI)

Where to use Python

- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming

Where to use Python

- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming
- Database (DB) programming

Where to use Python

- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming
- Database (DB) programming
- Text data processing

Where to use Python

- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming
- Database (DB) programming
- Text data processing
- Distributed processing

Where to use Python

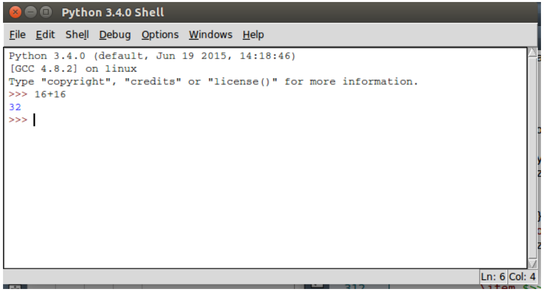
- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming
- Database (DB) programming
- Text data processing
- Distributed processing
- Numerical operations

Where to use Python

- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming
- Database (DB) programming
- Text data processing
- Distributed processing
- Numerical operations
- Graphics so on...

Introduction to Python

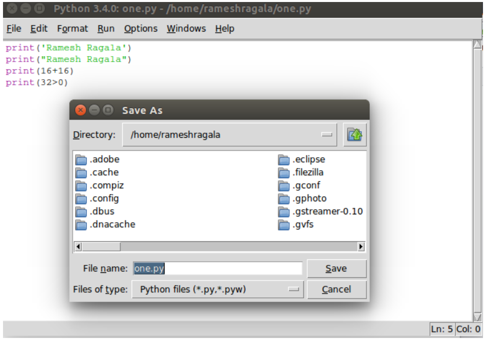
- Two ways to use python interpreter
  - 1. Interactive Mode
  - 2. Script Mode
- Interactive Mode:
  - we have to type python programs → Interpreter displays the result.
  - >>> 16+16
    32
  - The shell prompt, >>>, is the prompt the interpreter uses to indicate that it is ready.



```
Python 3.4.0 (default, Jun 19 2015, 14:18:46)
[GCC 4.8.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 16+16
32
>>> |
```

Introduction to Python

- Script Mode:
  - We can store code in a file and we can use interpreter to execute the content of the file $\rightarrow$ script.
  - Every python script saved with extension .py
  - Testing small pieces of code $\rightarrow$ Interactive Mode is good
    - Type and execute the piece of code immediately.
  - More number of lines of code $\rightarrow$ Script Mode
    - We are able to save the code
    - Editing and Execution of the code will be support in future also.


Python 3.4.0: one.py - /home/rameshragala/one.py

```
print('Ramesh Ragala')
print("Ramesh Ragala")
print(16+16)
print(32>0)
```

Identifiers in Python

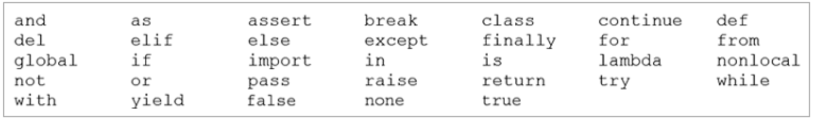
- **Identifier:**
  - It is a sequence of one or more characters used to name a given program element.
  - Example: line, salary, ragala10, VIT_UNIVERSITY.
- **Rules for Identifier**
  - Python is Case Sensitive. Good is different from good.
  - Identifiers may contains Letters and Digits. $\rightarrow$ can not start with digit
  - The special underscore character can also be used. $\rightarrow$ readability of long identifier names.
  - **Spaces are not allowed as part of an identifier**
  - The underscore characters not be used as the first character.

Identifiers in Python

Valid Identifiers	Invalid Identifiers	Reasons for Invalid
totalsales	'totalsales'	Quotes are not allowed
totalsales	total sales	Space is not allowed
salesfor2010	2010sales	Can not begin with a digit
sales_for_2010	_2010sales	Should not begin with underscore

Keywords in Python

- **Keywords:**
  - It is an identifier that has pre-defined meaning in a programming language.
  - So Keywords can not be used for regular Identifiers.
  - If we use Keywords as identifiers in program $\rightarrow$ syntax errors
  - Example: >>> and = 10
    SyntaxError: invalid Syntax



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		

Values and Types

- **Value:**
  - It is one of the basic things a program works with, like a letter or a number.
  - Examples: 1, 2 and 'Hello World'
  - 1 and 2 are belongs to Integer Category and 'Hello World' is a String.
  - If we want to know the type of the values:
    >>> type('Hello, World') $\rightarrow$ <class,'str'>.

Examples

>>>type(17) $\rightarrow$ result ???
>>>type(3.2) $\rightarrow$ output ???
>>>type('17') $\rightarrow$ output ???
>>>type("17") $\rightarrow$ Output ??
>>>type('3.2') $\rightarrow$ Output??
>>>**1,00,000** What would be the Answer $\rightarrow$ Semantic Error.
result is (1,0,0) $\rightarrow$ interpreter as comma separated sequence of integers

Variables in Python

- **Variable:**
  - It a name or identifier that refers to a value.
  - An Assignment Statement creates new variables and gives them new values:
  - >>> ram = "VIT University Chennai Campus is near Kandigai"
  - >>> n = 16
  - >>> pi = 3.1415
  - **State Diagrams or variables are discuss in class**
  - The type of a variable is the type of the value it refers to.
  - >>>type(ram) $\rightarrow$ result???
  - Meaningful names can be chosen to describe variables.
  - Variable names can be arbitrarily long.
  - Variable names contain both letters and numbers, but they have to begin with a letter.

Variables Examples

Example

```
a = 0
b = 2
print(a+b)

# other code

m ='0'
j='2'
print(m+j)

# other Code

k =" 0"
l=2
print(int(k)+l)
```

Variables Examples

Example

```
str = '␣My␣Name␣is␣Ramesh'
print(str)
```

- **OUTPUT:??**

Example

```
str1 = 'How'z life'
print(str1)
```

- **OUTPUT:??**

Example

```
str2 = "How'z␣life"
print(str2)
```

- **OUTPUT:??**

Comments in Python

- Adding notes to our program is good idea. → more readability
- These Notes are called Comments.
- Comments in Python are two types: 1. Single Line Comments 2. Multiline Comments
- 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 are most useful when they document non-obvious features of the code.

Example-1

```
# compute the percentage of the hour that has elapsed
>>>percentage = (minute * 100) /60
```

Example-2

```
>>>percentage = (minute * 100) /60 # Percentage of an Hour
```

Comments in Python

- **Multiline Comments:**
  - It can be specified with triple-quoted strings.

Example

```
'''
This is Ramesh ragala
Working in VIT University
'''
print('GHK')

"""
This is also Ramesh ragal
dfd
"""
print(16+16)
```

Literals in Python

- **Literal:**
  - Literals are notations for constant values of some built-in types.
  - It is a sequence of one or more characters that stands for itself
  - Different types of Literals used in Python:
    - String and Bytes Literals
    - Numerical Literals

    1. Integer Literals
    2. Floating Point Literals
    3. Imaginary Literals

Literals

- **String Literal:**
  - It represents a sequence of characters.
  - Example: 'Hello', ' VIT', "Chennai, Vellore-600127"
  - In Python, string literals may be delimited (surrounded) by a matching pair of either single (') or double (") quotes.

Example

```
print("Welcome␣to␣Python")
```

'A'	A string consisting of a single character
'abc@vit.ac.in'	A string consisting of non-letter characters
" how'z life "	A string consisting of a single quote characters
' '	A string containing a single blank character
''	the empty string

Introduction to Python

- >>> print('Hello') → check output
- >>> print('Hello") → check output
- >>> print('Let's Go') → Check output
- >>> print("Hello") → check output
- >>> print("Let's Go!')→ check output
- >>> print("Let's Go!")→ check output

Literals

- **Numeric Literals:**
  - There are three types.

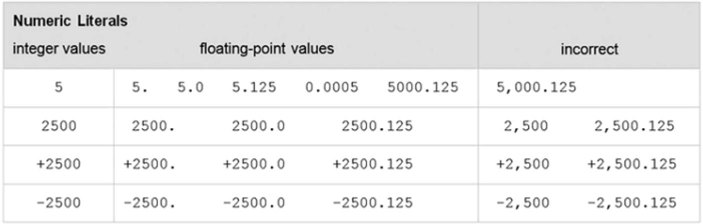
    1. Integer Literals:
       - A numeric literal is a literal containing only the digits 0-9, an optional sign character (1 or 2),and a possible decimal point. (e $\rightarrow$ exponential notation)
       - leading zeros in a non-zero decimal number are not allowed.
       - There is no limit for the length of integer literals apart from what can be stored in available memory.
       - Examples: 7 , 3 , 2147483647 etc

    2. Floating point Literals:
       - It contains decimal point.
       - the integer and exponent parts are always interpreted using radix 10.
       - Example: 3.18, 1e200, 0e0

    3.Imaginary Literals:
       - An imaginary literal yields a complex number with a real part of 0.0.
       - Complex numbers are represented as a pair of floating point numbers and have the same restrictions on their range.
       - Examples: 3.14j, 3.14e+10j

Introduction to Python

- Examples on Numerical Literals

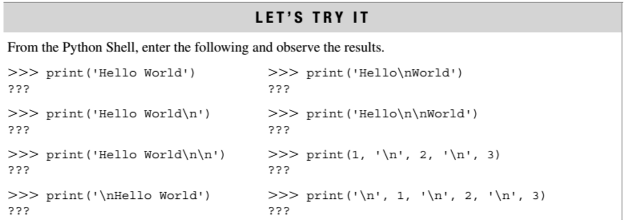


Numeric Literals		
integer values	floating-point values	incorrect
5	5. 5.0 5.125 0.0005 5000.125	5,000.125
2500	2500. 2500.0 2500.125	2,500 2,500.125
+2500	+2500. +2500.0 +2500.125	+2,500 +2,500.125
-2500	-2500. -2500.0 -2500.125	-2,500 -2,500.125

- >>> 1024 → check output
- >>> -1024 → check output
- >>> .1024 → check output
- >>> 1,024 → check output
- >>> 0.1024 → check output
- >>> 1,024.56 → check output

Introduction to Python

- **Control Characters:**
  - Special characters that are not displayed on the screen. $\rightarrow$ 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 $\rightarrow$ the newline control character, used to begin a new screen line.



LET'S TRY IT

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

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

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

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

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

Introduction to Python

- **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
    1. Boolean Values
    2. None
    3. Numbers
    4. Strings
    5. Tuples
    6. Lists
    7. Sets

Introduction to Python

- **Boolean values:**
  - Primitive datatype having one of two values: True or False.
  - Some common values that are considered to be True or False.
  - Find the outputs for the following

Examples

```
print(bool(True))
print(bool(False))
print(bool("ramesh"))
print(bool(""))
print(bool(''))
print(bool(0))
print(bool(3))
print(bool(None))
print(True + 25)
print(False + 25)
print(bool.__bases__)
```

Introduction to Python

- **None:**
  - None is a special value.
  - It is a value that indicates no value.
  - Can be used to check for emptiness.
  - Try it: type(None) $\rightarrow$ Output???
  - >>> x = None
  - >>> help(x) $\rightarrow$ Output???

Introduction to Python

- Types of Numbers supported by Python
  1. Integers
  2. Floating Point Numbers
  3. Complex Numbers
  4. Fractional Numbers
- **Integers:**
  - Integers have no fractional part in the number.
  - Integer type automatically provides extra precision for large numbers like this when needed (different in Python 2.X)
  - `>>> a = 10`
  - `>>> b = a`
- **Binary, Octal and Hexadecimal Numbers:**
  - 0b1, 0b10000, 0b11111111 → binary literals → base - 2 and digits are 0 and 1 only.
  - 0o1, 0o20 and 0o377 → Octal literals → base - 8 and digits are 0 to 7 only.
  - 0x01, 0x10 and 0xFF → Hexadecimal literals → base - 16 and digits are 0 to 9 and A to F.

Introduction to Python

- **Conversion between different bases**
- Provides built-in functions that allow you to convert integers to other base's digit strings.
  - oct(64), hex(64) and bin(64) $\rightarrow$ check outputs
  - 0o100, 0x40, 0b1000000 $\rightarrow$ check outputs
  - >>> X = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF $\rightarrow$ check output
  - >>> oct(X) $\rightarrow$ check output
  - >>> bin(X) $\rightarrow$ check output
- **Floating point Numbers:**
  - Number with a fractional part
  - >>> 3.1415 * 2 $\rightarrow$ check output

Introduction to python

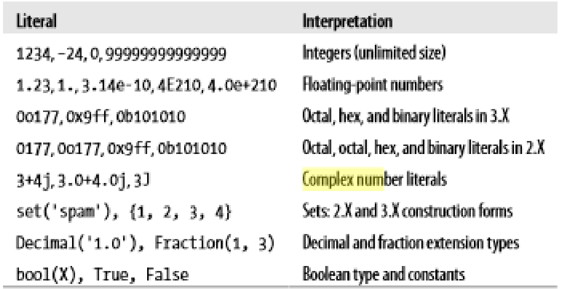
- **Arithmetic overflow**
  - A condition that occurs when a calculated result is **too large** in magnitude (size) to be represented.
  - Example: >>> 1.5e200 * 2.0e210 → check result
  - This results in the special value **inf** ("infinity") rather than the arithmetically correct result 3.0e410, indicating that arithmetic overflow has occurred.
- **Arithmetic underflow**
  - A condition that occurs when a calculated result is **too small** in magnitude to be represented.
  - Example: >>> 1.0e-300 / 1.0e100 → check output
  - This results in 0.0 rather than the arithmetically correct result 1.0e-400, indicating that arithmetic underflow has occurred.

Introduction to python

- **Complex Numbers:**
  - A complex number consists of an ordered pair of real floating point numbers denoted by a + bj → a and b →real part and imaginary part of the complex number.
  - complex(a) → convert to a complex number with real part as a and imaginary part as zero.
  - complex(a,b) → convert to a complex number with real part as and imaginary part as b.⇒ a and b are numeric expressions.
  - Examples:
    - >>>1j * 1J → check output → (-1+0j)
    - >>>2+1j * 3 → check output → (2+3j)
    - >>>(2+1j)*3 → check output → (6+3j)
    - >>>A = 1+2j; B=3+2j → check output → Mutpile statements can be given in same line using semicolon → check A and B
    - >>> print(A.real) → check output → prints real part of the number.
    - >>> print(A.imag) → check output → prints imaginary part of the number.
    - >>> print(A.imag+3) → check output → doing the operations with part of complex number

Introduction to Python

Table 5-1. Numeric literals and constructors



Literal	Interpretation
`1234, -24, 0, 99999999999999`	Integers (unlimited size)
`1.23, 1., 3.14e-10, 4E210, 4.0e+210`	Floating-point numbers
`0o177, 0x9ff, 0b101010`	Octal, hex, and binary literals in 3.X
`0177, 0o177, 0x9ff, 0b101010`	Octal, octal, hex, and binary literals in 2.X
`3+4j, 3.0+4.0j, 3J`	Complex number literals
`set('spam'), {1, 2, 3, 4}`	Sets: 2.X and 3.X construction forms
`Decimal('1.0'), Fraction(1, 3)`	Decimal and fraction extension types
`bool(X), True, False`	Boolean type and constants

- **Repeated Print:**
  - >>> print(a*10) → check output → prints 'a' ten times.
  - >>> print(”*10) → check output → prints new line character ten times.

Introduction to Python

- **Input and Output Functions in Python:**
  - **input** $\rightarrow$ Basic input function in python
  - Example: **nameofstudent = input('Enter your Name')**
  - **print** $\rightarrow$ Basic output function in python
  - Example: **print("Student name", nameofstudent)**
- **Type Conversion**
  - input() is used to read the data in string format
  - type conversion **int()** is used to convert string into integer
  - type conversion **float()** is used to convert string into float
  - Example: **amount = int(input("Enter amount:"))**
  - Another approach: **amount = input("Enter amount:")**
    amount = int(amount)
  - Example: **amount = float(input("Enter amount:"))**

Introduction to Python

Statement	Type
spam = 'Spam'	Basic form
spam, ham = 'yum', 'YUM'	Tuple assignment (positional)
[spam, ham] = ['yum', 'YUM']	List assignment (positional)
a, b, c, d = 'spam'	Sequence assignment, generalized
a, *b = 'spam'	Extended sequence unpacking (Python 3.X)
spam = ham = 'lunch'	Multiple-target assignment
spams += 42	Augmented assignment (equivalent to spams = spams + 42)

Introduction to Python

- **Range:**
  - >>> a,b,c = range(1,4)
  - >>> a → check output → 1
  - >>> b → check output → 2
  - >>> c → check output → 3
  - >>> k = "vit"; k += "VIT" → check output → Implied concatenation
- Assignment is more powerful in python:
  - >>> A = 10; B = 20
  - >>> A, B = B, A → check output → swapping
  - another way (general way:)
  - >>> T=A; A=B; B=T → check output → swapping

Introduction to Python

- Operators are special symbols in Python that carry out arithmetic or logical computation.
- The value that the operator operates on is called the operand.
- Operators and Expressions in Python:
  - **Basic Arithmetic Operations in Python**

command	Name	Example	Output
+	Addition	4 + 5	9
-	Subtraction	8 - 5	3
*	Multiplication	4 * 5	20
/	True Division	19 / 3	6.333
//	Integer Division	19 // 3	6
%	Remainder	19 % 3	1
**	Exponent	2 ** 4	16

Introduction to Python

- Operators and Expression in Python
  - **Order of Operations:**
    - 1. Parentheses ()
    - 2. Exponents **
    - 3. Multiplication *, Division / and remainder %
    - 4. Addition + and subtraction -

Operator	**Operation**	**Precedence**	**Associativity**
()	Parentheses	0	Left to Right
**	Exponentiation	1	Right to Left
*	Multiplication	2	Left to Right
/	Division	2	Left to Right
//	Integer Division	2	Left to Right
%	Remainder	2	Left to Right
+	Addition	3	Left to Right
-	Subtraction	3	Left to Right

- **Operators and Expressions in Python**
  - **Examples:**
    - >>> 1 + 2 * 3 → check output → Priority
    - >>> (1 + 2) * 3 → check output → Priority
    - >>> 4 - 40 - 3 → check output → Associativity
    - >>> 4 -(40 - 3) → check output → priority
    - Check the following outputs in interactive mode of python
    - word = 'word' → check output
    - sentence = "This is Sentence" → check output
    - paragraph = """ This is a paragraph. it is made of multiple line""" → check output → legal
    - name = int(input('Enter name')) → given name as vit → check output → reason

Introduction to Python

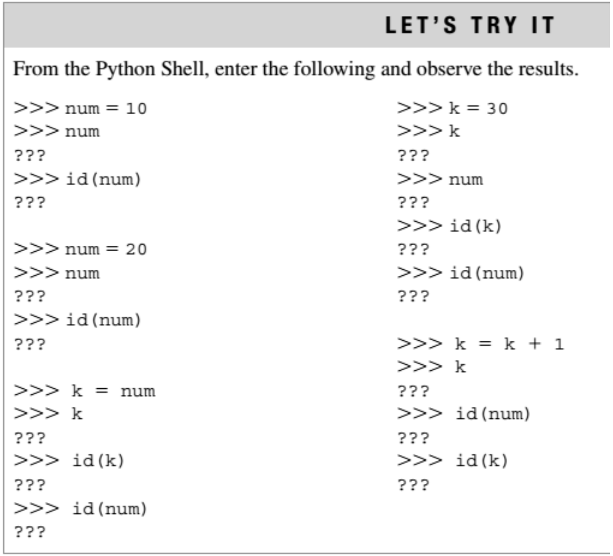
- **Built - in Format Function:**
  - Built-in Function is used to produce a numeric string version of a value containing a specific number.
  - >>> 12/5 → check output → 2.4
  - >>> format(12/5, '.2f') → check output
  - >>> 5/7 → check output
  - >>> format(5/7, '.2f') → check output → '.2f' rounds the result to two decimal places of accuracy in the string produced.
  - For very large and very small values 'e' can be used as a format specifier.
  - >>> format(2 ** 100, '.6e') → check output
  - >>> format(11/12, '.2f') → check output
  - >>> format(11/12, '.3f') → check output
  - >>> format(11/12, '.2e') → check output
  - >>> format(11/12, '.3E') → check output

Introduction to Python

- **Python: Dynamic Type Language**
  - Same variable can be associated with values of different type during program execution.
  - Example
    - `>>> var10 = 10`
    - `>>> var10 = 10.24`
    - `>>> var10 = ' VIT '`
  - It is also very dynamic as it rarely uses what it knows to limit variable usage

Introduction to Python

- **Examples**



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)
???
```

Introduction to Python

- **Bitwise Operators:**
  - Manipulation can be done at bit level.
  - It treats the integers as strings of binary bits.
  - These operators are useful in network packets, serial port and binary packet data.
  - >>> x = 1 → decimal 1 is 0001 in bits
  - >>> x << 2 →check output → Shift left 2 bits → 0100
  - Three different bitwise operators:
    - 1. Bitwise AND Operator
    - 2. Bitwise OR Operator
    - 3. Bitwise NOT Operator
  - >>> x | 2 → check output → Bitwise OR Operator
  - >>> x & 2 → check output → Bitwise AND Operator
  - >>> x = 0b0001 → >>> bin(x << 2) → check output
  - >>> bin(x | 0b010) → check output
  - >>> bin(x & 0b1) → check output

Introduction to Python

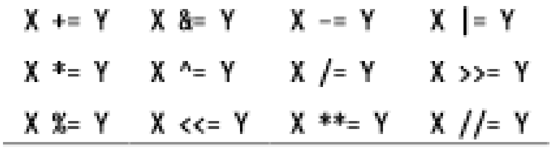
- **Logical Operators:**
  - Assume a = 10 and b = 20

Logical Operator	**Description**	**Example**
and	If both the operands are true then condition becomes true	(a and b) is true
or	If any of the two operands are non-zero then condition becomes true	(a or b) is true
not	Used to reverse the logical state of its operand	Not(a and b) is false

Introduction to Python

- **Python is strongly typed language:**
  - interpreter keeps track of all variables types
  - Check type compatibility while expressions are evaluated
  - >>> 2 + 3 → check output
  - >>> 'two' + 1 → check output
- **Shorthand Assignment operators**

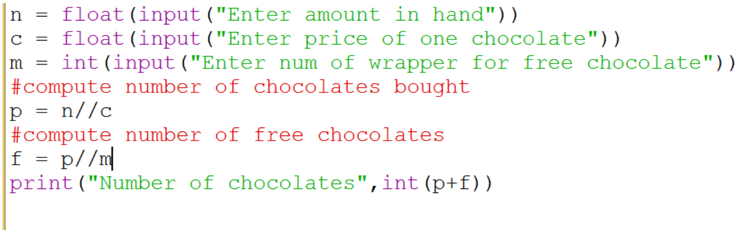
Table 11-2. Augmented assignment statements



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

Introduction to Python

- Python program for Bob Problem



```python
n = float(input("Enter amount in hand"))
c = float(input("Enter price of one chocolate"))
m = int(input("Enter num of wrapper for free chocolate"))
#compute number of chocolates bought
p = n//c
#compute number of free chocolates
f = p//m
print("Number of chocolates",int(p+f))
```

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 and HRA are calculated as certain % of Basic pay(For example, 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. Propose a computerized solution for the above said problem.

- We know the **PAC chart** and **Flowchat**

code

```
#Enter the basic pay
bp=float (input('Enter the basic pay:'))
# net pay caluclation
netpay =bp + (bp*0.8) + (bp*0.3) - (bp*0.12)
# display net salary
print ('Net pay :',netpay)
```

Introduction to Python

- **lambda Operator:**
  - The lambda operator or lambda function is a way to create small anonymous functions → i.e. functions without a name.
  - Example
  - `>>> ftoc = lambda f:(f-32)*5.0/9`
  - `>>> print(ftoc(104))` → check output

Thank you