Problem Solving and Programming

Prof. Tulasi Prasad

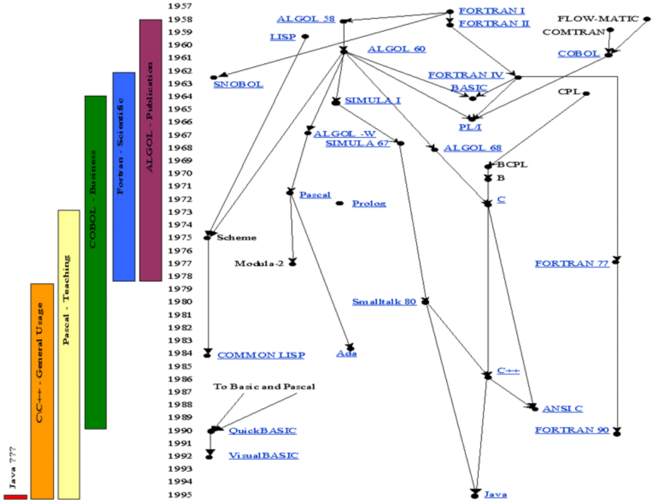
July 30, 2019

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. ⇒ 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.

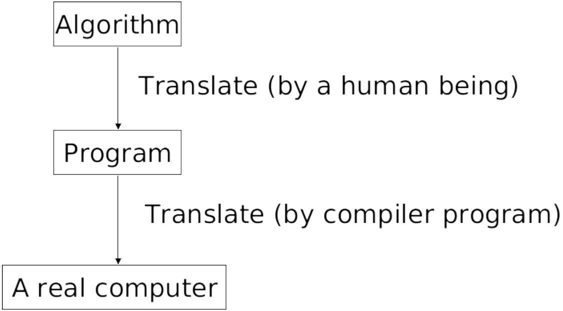
Examples of Programming Language

- History of Programming Languages



Need of Programming Language

- Algorithm to Hardware



Algorithm

Translate (by a human being)

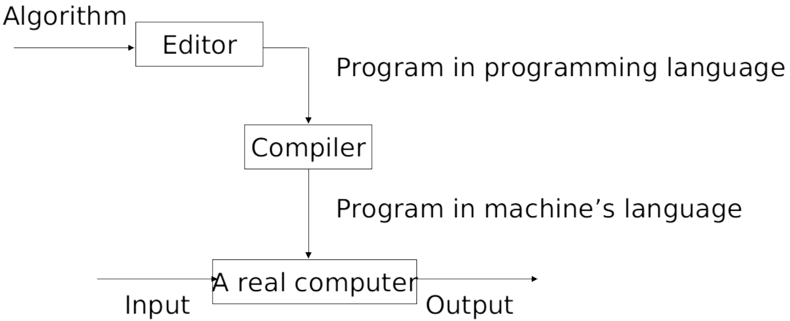
Program

Translate (by compiler program)

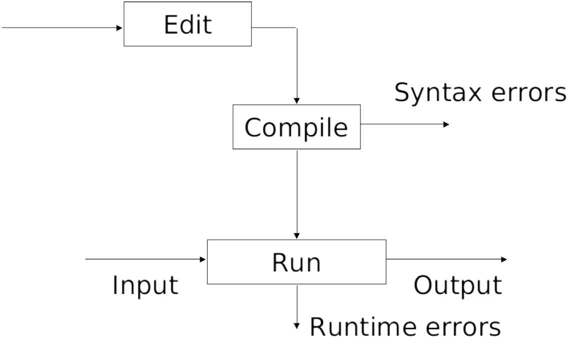
A real computer

Need of Programming Language

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

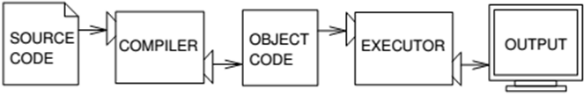

Algorithm
Editor
Program in programming language
Compiler
Program in machine's language
A real computer
Input
Output

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



Compiler

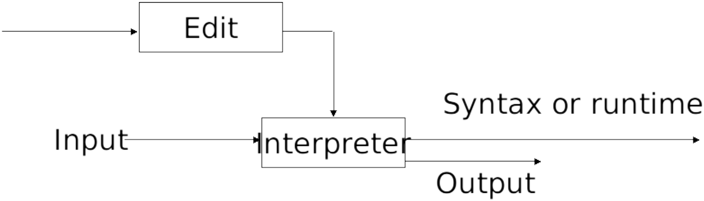
- It is a program that converts a program written in a programming language into a program in the native language, called machine language, of the machine that is to execute the program.
- It reads the program and translates it completely before the program starts running.
- High Level Program is called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

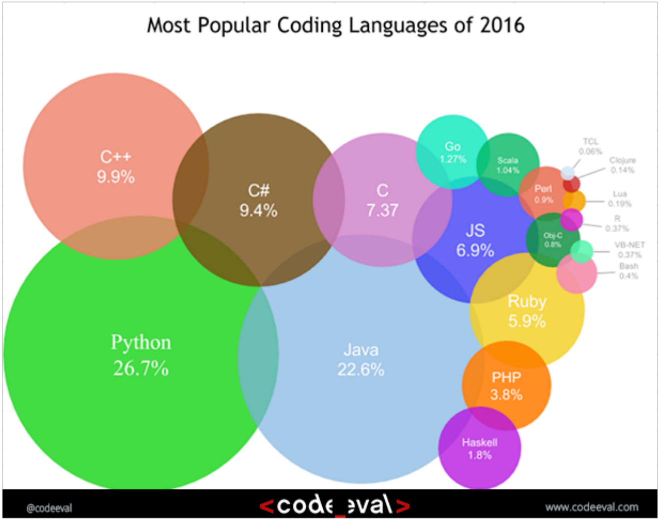
- Interpreter
  - It takes our program's statements at a time (one by one) and executes a corresponding set of machine instructions.
  - It is alternative to compiler.
  - The processing is slow.



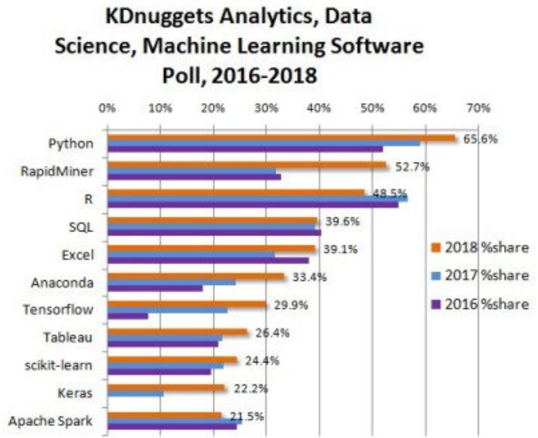
Types of Error in Program Language

- 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.

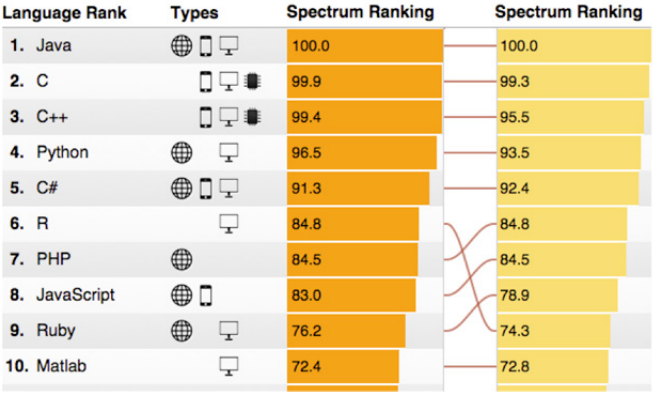
- Python Ranking



- Python Ranking

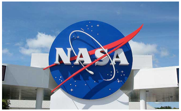


- Python Ranking according to IEEE



Language Rank	Types	Spectrum Ranking	Spectrum Ranking
1. Java	🌐 📱 🖥	100.0	100.0
2. C	📱 🖥 ▣	99.9	99.3
3. C++	📱 🖥 ▣	99.4	95.5
4. Python	🌐 🖥	96.5	93.5
5. C#	🌐 📱 🖥	91.3	92.4
6. R	🖥	84.8	84.8
7. PHP	🌐	84.5	84.5
8. JavaScript	🌐 📱	83.0	78.9
9. Ruby	🌐 🖥	76.2	74.3
10. Matlab	🖥	72.4	72.8

Python Users



Walt Disney
FEATURE ANIMATION
ANIMATION TRAINING PROGRAMS

NOKIA
Connecting People

- 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. → 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. → 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: 3.7.3 and 2.7.12

Introduction to Python

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

Python Language Properties

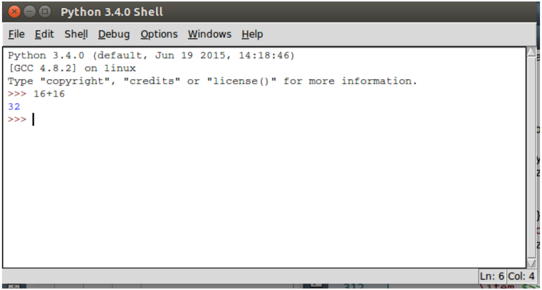
- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure

Where to use Python

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

Introduction to Python

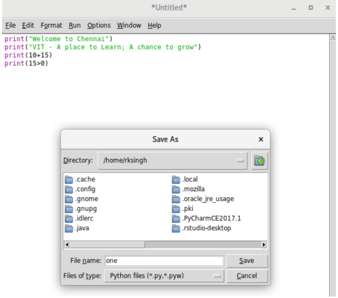
- Two ways to use python interpreter
  - 1. Interactive Mode
  - 2. Script Mode
- Interactive Mode:
  - we have to type python programs → Interpreter displays the result.
  - >>> 16+16
    32
  - The shell prompt, >>>, is the prompt the interpreter uses to indicate that it is ready.



```
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 → script.
  - Every python script saved with extension .py
  - Testing small pieces of code → Interactive Mode is good
    - Type and execute the piece of code immediately.
  - More number of lines of code → Script Mode
    - We are able to save the code
    - Editing and Execution of the code will be support in future also.



```
print("Welcome to Chennai")
print("VIT - A place to Learn; A chance to grow")
print(10+15)
print(15>0)
```

Directory: /home/rksingh

File name: one

Files of type: Python files (*.py,*.pyw)

- In Python, string literals may be delimited (surrounded) by a matching pair of either single (') or double (") quotes.
- Problems that can be solved by Sequential Algorithms

EXAMPLE:

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?

PAC Chart:

Input	Processing	Output	Alternative Solutions
MoneyInHand, CostOfChocolate, WrapperOffer	NumberOfChocolates =Chocolategotby money + Chocolategotbyreturningwrapper	NumberOfChocolate	-

Introduction to Python

same Example with values

Bob has Rs. 100 in his pocket. The price of each chocolate is Rs. 5. The store offers a free chocolate for every 4 wrappers he gives to the store, and he gives all wrappers. This offer is available only once. How many chocolates does Bob get to eat?

simple procedure

```
n = 100
c = 5
m = 4
p = n/c
f = p/m
print(”Number of chocolates::”, p+f)
```

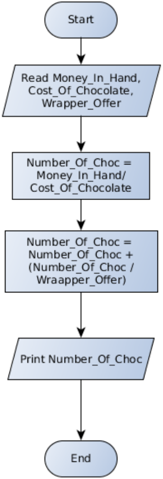
- **Type above procedure in python console → check output**
- so we need to write generalized procedure for the above problem

Pseudo Code

```
STEP -1: READ MoneyInHand and CostOfChocolate
STEP -2: COMPUTE NumberOfChocolate as MoneyInHand or CostOfChocolate
STEP -3: NumberOfChocolate = NumberOfChocolate +(NumberOfChocolate or WrapperOffer)
STEP -4: PRINT NumOfChocolates
```

Introduction to Python

- Flow Chart for the problem:


Start
Read Money_In_Hand, Cost_Of_Chocolate, Wrapper_Offer
Number_Of_Choc = Money_In_Hand/ Cost_Of_Chocolate
Number_Of_Choc = Number_Of_Choc + (Number_Of_Choc / Wraapper_Offer)
Print Number_Of_Choc
End

Python Code

```
n=int(input("Enter␣the␣money␣in␣Hand"))
c=int(input("Enter␣the␣Unit␣Price␣of␣Chocolate"))
m=int(input("Enter␣Strore␣offer"))

p = n/c
f = p/m
print( Number of chocolates::     , p+f)
```

Knowledge Required

- Following knowledge is required in Python to write a code to solve the above problem
    - Read input from user
    - Data types in Python
    - Perform arithmetic calculations
    - Write output

Identifiers in Python

- **Identifier:**
  - It is a sequence of one or more characters used to name a given program element.
  - Example: line, salary, ram10, VIT_UNIVERSITY.
- **Rules for Identifier**
  - Python is Case Sensitive. Good is different from good.
  - Identifiers may contains Letters and Digits. → can not start with digit
  - The special underscore character can also be used. → readability of long identifier names.
  - **Spaces are not allowed as part of an identifier**
  - The underscore characters not be used as the first character.

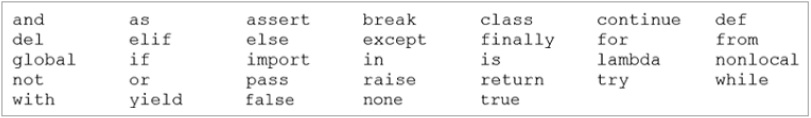
Identifiers in Python

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 → 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’) → <class,’str’>.

Examples

\>>>type(17) → result ???

\>>>type(3.2) → output ???

\>>>type(’17’) → output ???

\>>>type(”17”) → Output ??

\>>>type(’3.2’) → Output??

>>>**1,00,000** What would be the Answer → Semantic Error.

result is (1,0,0) → interpreter as comma separated sequence of integers

Variables in Python

- **Variable:**
  - It is 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
  - The type of a variable is the type of the value it refers to.
  - >>>type(ram) → 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␣Sachin'
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

```
'''
VIT Chennai Campus
VIT - A place to learn; A Chance to grow
'''
print('GHK')

"""
You have chosen yourself to sttudy at VIT Chennai
VIT - A place to learn; A Chance to grow
"""
print(16+16)
```

Literals in Python

- **Literal:**
  - Literals are notations used 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-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

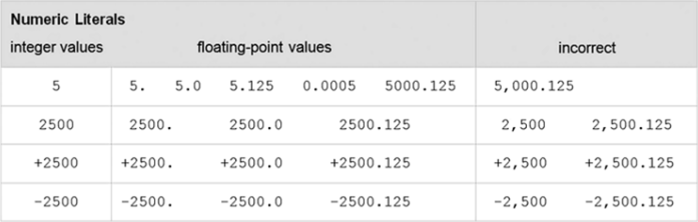
Literals

- **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 → 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

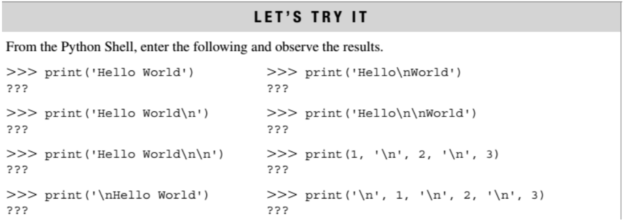
- Examples on Numerical Literals



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

- **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 → 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)
???                                  ???
```

- **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

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

- **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) → Output???
  - $>>>$ x $=$ None
  - $>>>$ help(x) → Output???

- 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

Binary, Octal and Hex Literals

- 0b1, 0b10000, 0b11111111 # Binary literals: base 2, digits 0-1
- 0o1, 0o20, 0o377
- # Octal literals: base 8, digits 0-7
- 0x01, 0x10, 0xFF # Hex literals: base 16, digits 0-9/A-F
- (1, 16, 255)

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')
- These literals can produce arbitrarily long integers

Numbers can be very long

- >>> X = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF
- >>> X
  5192296858534827628530496329220095
- >>> oct(X)
  '0o17777777777777777777777777777777777777
- >>> bin(X)
  '0b111111111111111111111111111111111111111111111111111111111
- ...and so on... 11111

Floating Point Numbers

- Number with fractional part
- >>> 3.1415 * 2
- >>> 6.283

Numeric Literals and Constructors

Literals	Interpretation
1234 , 24 , 0 , 9999999999	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, 0x9fff, 0b1010101	Octal,octal, Hex and Binary literals in 2.X
3+4j, 3.0+4.0j, j	Complex number literals
set('spam'), 1,2,3,4	Set: 2.X and 3.X Construction form
Decimal('1.0'), Fraction(1,3)	Decimal and Fraction extension type
bool(x), True, False	Boolean type constants

Arithmetic overflow

- A condition that occurs when a calculated result is too large in magnitude (size) to be represented,

  ```
  >>>1.5e200 * 2.0e210
  >>> inf
  ```

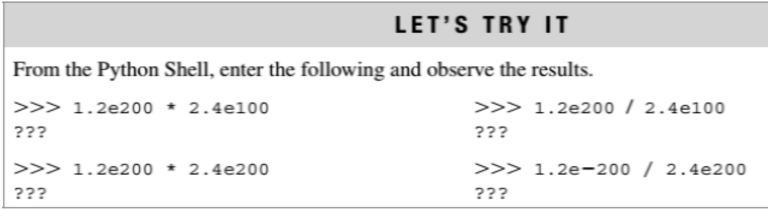
  This results in the special value inf (infinity) rather than the arithmetically correct result 3.0e410, indicating that arithmetic overflow has occurred.

Arithmetic underflow

- a condition that occurs when a calculated result is too small in magnitude to be represented,

```
>>> 1.0e-300 / 1.0e100
>>> 0.0
```

- This results in 0.0 rather than the arithmetically correct result 1.0e-400, indicating that arithmetic underflow has occurred.



LET'S TRY IT

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

```
>>> 1.2e200 * 2.4e100                    >>> 1.2e200 / 2.4e100
???                                      ???

>>> 1.2e200 * 2.4e200                    >>> 1.2e−200 / 2.4e200
???                                      ???
```

Arithmetic overflow occurs when a calculated result is too large in magnitude to be represented.
Arithmetic underflow occurs when a calculated result is too small in magnitude to be represented

Repeated Print

```
>>> print('a' *15)
# prints 'a' fifteen times
>>> print('\n'*15)
# prints new line character fifteen times
```

Complex Numbers

- A complex number consists of an ordered pair of real floating point numbers denoted by a + bj, where a is the real part and b is the imaginary part of the complex number.
- **complex(x)** to convert x to a complex number with real part x and imaginary part zero
- **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y.
- x and y are numeric expressions

Complex Numbers

```
>>> 1j * 1J
(-1+0j)
>>> 2 + 1j * 3
(2+3j)
>>> (2 + 1j) * 3
(6+3j)
```

- A = 1+2j; B=3+2j
- # Multiple statements can be given in same line using semicolon
- C = A+B; print(C)
- print(A.real)
  # prints real part of the number
- print(A.imag)
  # prints imaginary part of the number
- print(A.imag+3)
  # Can do operations with part of complex number

Input and output function

Input function : input
Basicpay = input('Enter the Basic Pay: ')
Output function : print
print('Hello world!')
print(Net Salary', salary)

By default...

Input function reads all values as strings, to convert then to integers and float, use the function int() and float()

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)
```

Alternatively

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

Assignment Statement

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)

Range

```
>>>a,b,c = range(1,4)
>>> a
1
>>> b
2
>>> S = "spam"
>>> S+= "SPAM"     # Implied concatenation
>>> S
'spamSPAM'
```

Assignment is more powerful in Python

```
>>> nudge = 1
>>> wink = 2
>>> nudge, wink = wink, nudge
# Tuples: swaps values
# Like T = nudge; nudge = wink; wink = T
>>> nudge, wink
(2, 1)
```