

- Ctrl + = increase the size of font of eclipse editor
  - Ctrl - = decrease the size of font of eclipse editor
  - IDLE is a acronym for **Integrated Development** and Learning Environment
  - IDE is a acronym which stands for **Integrated Development Environment**.
- 

- *Author Guido van Rossum says IDLE stands for "Integrated DeveLopment Environment".*
- *Since van Rossum named the language Python partly to honor British comedy group [Monty Python](#).*
- *The name IDLE was probably also chosen partly to honor [Eric Idle](#), one of Monty Python's founding members.*

- Applications developed using  
Python

- Python has been used to create some of the most popular websites in the world of internet such as:
- 1. [Google](#): One of the most popular search engines in the world has been built using Python. Python allows Google to switch the traffic and figure out the requirements of search.
- 2. [YouTube](#): Python has been the driving force behind YouTube, a website used by millions for downloading and uploading videos of all hues and sizes. The website has been coded in a way which makes it easier and extremely interactive for the user.
- 3. [Quora](#): It's a portal where you get your answers. You can post a question and you can get an answer from any part of the world. Quora's language programming has been developed using Python's framework.
- 4. [Dropbox](#): Many of our choices to store our data are going online. We create a document, we save it, and we share it. All of this is done online using Dropbox. It is an ideal way to preserve your documents online. This file hosting service has also been created using Python.

- [Yahoo!](#):Google's biggest competitor in the search engine criteria. Yahoo and many of its subsidiaries, including Yahoo Maps, have been designed using Python.
- [Instagram](#):Uploading and sharing photos has never been this exciting. Instagram has revolutionized the way pictures and videos are shared. The popular picture sharing website also relies heavily on Python for many of its functionalities, including the video sharing service.
- [Tornado: Facebook's Real-Time Web Framework for Python](#)
- **Games:**
- **Civilization IV, Battlefield 2, World of Tanks**

- Virtual Personal Assistants
- Social Media Services
- Online Customer Support
- Online Fraud Detection
- Product Recommendations
- Refining Search Engine Results
- Fighting Web Spam
- Automatic Translation
- Video Surveillance
- Predicting Music Choices
- Drug Discovery & Disease Diagnosis
- Face Recognition
- Pricing Insurance Plans
- Autonomous, Self-Driving Cars

# Applications of Machine Learning with Python

# Why Python

- **Easy – to – learn.**
- Code is **3-5 times shorter than Java.**
- **5-10 times shorter than C++.**
- It allows you to express very powerful ideas in **very few lines of code** while being very readable.
- Therefore **maintainability is high.**
- **Stepping Stone** to Programming universe.
- Python code is often said to be **almost like pseudocode.**

# Python v/s Java

A simple Program to print "Hello World"

Java Code	Python Code
<pre>public class HelloWorld {     public static void main(String args [] )     {         System.out.println ("Hello World!")     } }</pre>	<pre>print ("Hello World!")</pre>

- PYTHON WAS **NOT MADE TO BE FAST.**
- BUT TO **MAKE DEVELOPERS/MAINTAINERS FAST.**



# Evolution of Python

- **Guido van Rossum** is a Dutch programmer is the author of the Python programming language developed it in 1990 at national research institute for Mathematics and Computer Science
- Named after a circus show “Monty Python Show”
- Derives its features from many languages like JAVA, C++, ABC, C, Modula-3, Smalltalk, Unix Shell and other scripting languages.
- Available GNU General Public License(GPL)- Free and open-source software

## Evolution of Python

- **Guido Van Rossum** developed Python in **early 1990s** at National Research Institute for Mathematics and Computer Science, Netherlands.
- Named after a circus show Monty Python show.
- Derives its features from many languages like **Java, C++, ABC, C, Modula-3, Smalltalk, Algol-68, Unix shell** and other scripting languages.
- Available under the GNU General Public License (GPL) – Free and open-source software.

# Google-Fuchsia

- New open source operating system
- Google is dumping Linux and the GPL and most likely Java and all the problems they have had with Oracle.

**Google's Fuchsia OS Gets A New Logo**



- The actual operating system will be very different in how Android was designed, but Google worked on Android, therefore many things will also be the same/similar.
- **Fuchsia** is **written in Go**, Rust, Dart, C, C++ and **Python**, unlike Android, which is mainly written in Java.

# Facts about Python

- The Python programming language can be written in any language.
- The "standard" Python interpreter is written in C (also known as CPython).
- Most of the standard library that comes along with this version of Python is written in Python itself.
- Python depends on the **platform's C library**.

# Facts about Python contd..

- There are also implementations of Python in other languages:
- [Jython](#) is a version of Python designed to run on the Java platform, written in Java.
- [IronPython](#) is a version of Python running on the .NET platform, written in C#.
- [Stackless Python](#) is an alternative implementation of Python written in C, capable of using microthreads for parallelism.
- [PyPy](#) is a self-hosting (written in Python) JIT implementation of Python, which is capable of producing native machine code from Python code.

# Facts about Python

- Python's methodologies can be used in a **broad range of applications**.
- Bridging the Gap between abstract computing and real world applications.
- **Rising Demand** for Python Programmers.
- Google, Nokia, Disney, Yahoo, IBM use Python.
- **Not only a Scripting language, also supports Web Development and Database Connectivity.**

# Facts about Python contd..

1. **Python** is a **high-level**, interpreted, Interactive **dynamically typed**, multi paradigm programming language.
2. **Java** is **statically typed** language.
3. Open- Source, **Object - Oriented**, **procedural** and **functional**.
4. Very **rich scientific computing libraries**.
5. Well thought out language, allowing to write **very readable** and **well structured code**: we “code what we think”.
6. As an example, here is an implementation of the classic **quicksort algorithm in Python**:



# Quick Sort

- **def quicksort(arr):**
- **if len(arr) <= 1:**
- **return arr**
- **pivot = arr[len(arr) // 2]**
- **left = [x for x in arr if x < pivot]**
- **middle = [x for x in arr if x == pivot]**
- **right = [x for x in arr if x > pivot]**
- **return quicksort(left) + middle + quicksort(right)**
- **L=quicksort([3,6,8,10,1,2,1])**
- **print(l)**
- ***Out Put: "[1, 1, 2, 3, 6, 8, 10]"***

# The **Scientific Python** ecosystem

- Unlike Matlab, or R, Python does not come with a pre-bundled set of modules for scientific computing.
- Next slide lists the basic building blocks that can be combined to obtain a scientific computing environment.

# Core numeric libraries

- **Numpy**: numerical computing with powerful **numerical arrays** objects, and routines to manipulate them. <http://www.numpy.org/>
- **Scipy** : high-level numerical routines. Optimization, regression, interpolation, etc <http://www.scipy.org/>
- **Matplotlib** : 2-D visualization, “publication-ready” plots <http://matplotlib.org/>

# Domain-specific packages,

- **mayavi** for 3-D visualization
- **pandas, statsmodels, seaborn** for statistics
- **sympy** for symbolic computing
- **scikit-image** for image processing
- **scikit-learn** for machine learning
- **tensorflow, keras** for **deep learning**

# Built-in Data Types

- All **data types** in Python program are **represented by objects** and relationship among objects
- An object is an in-computer-memory representation of a value from a particular data type.

# Each object is characterized by its identity, type and value

- **Identity**: it uniquely identifies an object. It is a **location in the computer's memory** where the object is stored. An object's *identity* never changes once it has been created. Which can be **found out using a function: `id(x)`**
- **Type**: type of an object completely specifies its behavior-
  - (i) the set of values it might represent: **`type(x)`**
  - (ii) the set of operations that can be performed on it  
e.g. **`x.upper()`** converts string x to upper case letters
- **Value**: The value of an object is the data-type value that it represents

# Built-in Data Types

## Numbers:

- Python supports four different numerical types –
  1. **int** (signed integers) **e.g.** `a=234`
  2. **Float** (floating point real values) **e.g.** `a=23.456`
  3. **Complex** (complex numbers): `x + yj`
    - **e.g.** `Z=9.3 + 3.6j`
    - **`print(z, z.real, z.imag)`** will print
    - `(9.3 + 3.6 j)`, `9.3` and `3.6` respectively

```
x = 3
print( type(x) )      # Prints "<type 'int'>"
print (x)             # Prints "3"
print (x + 1)          # Addition; prints "4"
print (x - 1)          # Subtraction; prints "2"
print (x * 2)          # Multiplication; prints "6"
print (x ** 2)         # Exponentiation; prints "9"
x += 1
print (x)              # Prints "4"
x *= 2
print (x)              # Prints "8"
y = 2.5
print (type(y))        # Prints "<type 'float'>"
print (y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```



# String

- `s = "hello"`
- `print (s.capitalize())`     *# Capitalize a string; prints "Hello"*
- `print (s.upper())`     *# Convert a string to uppercase; prints "HELLO"*
- `print (s.rjust(7))`  
    *# Right-justify a string, padding with spaces; prints " hello"*
- `print (s.center(7))`  
    *# Center a string, padding with spaces; prints " hello "*
- `print (s.replace('l', '(ell)'))` )  
    *# Replace all instances of one substring with another;*  
    *# prints "he(ell)(ell)o"*
- `print (' worldw '.strip())`  
    *# Strip leading and trailing whitespace; prints "world"*
- `print ('wtwwwworldw'.strip('w'))`  
    *# Strip leading and trailing 'w'; prints "world"*

# Bool (for True and False values)

- `t = True`
- `f = False`
- `print (type(t))`    *# Prints "<type 'bool'>"*
- `print (t and f)`    *# Logical AND; prints "False"*
- `print (t or f)`    *# Logical OR; prints "True"*
- `print (not t )`    *# Logical NOT; prints "False"*
- `print (t != f)`    *# Logical XOR; prints "True"*

# Commenting Style in Python!

- **Types of Comments:**
- **A single line comment starts** with hash symbol **#** and end as the line ends.
- These lines are never executed and are ignored by the interpreter. e.g.
- **#** This is a single line comment
- **Multi line comments** *starts and ends* with
  1. **Triple single quotes '''** or
  2. **Triple """" double quotes.**
- Used for documentation.

# Multiline statements

- Python statements **always end up with a new line**,
- But it also allows **multiline statement using “ \” character at the end of line as shown below:**
- **result = (8+5)\* 2+\**
- **9/5**
- Statements which have **( ), [ ], { }** brackets and comma, **do not need any multiline character** to go to next line shown below:
- **customer\_details = [101, 'psit', 164, "Kanpur",  
'165', 498.24]**

- **Padding and aligning**  
**using**  
**% (old style)**  
**or**  
**.format() (New style)**

# Padding and aligning

- By default **values are formatted** to take up only as many **characters as needed** to represent the content.
- It is however also **possible to define** that a value should be **padded to a specific length**.
- The default alignment differs between **old** and **new style formatting**.
- The **old style defaults to right aligned** while for **new style it's left**.

# Padding and aligning using %

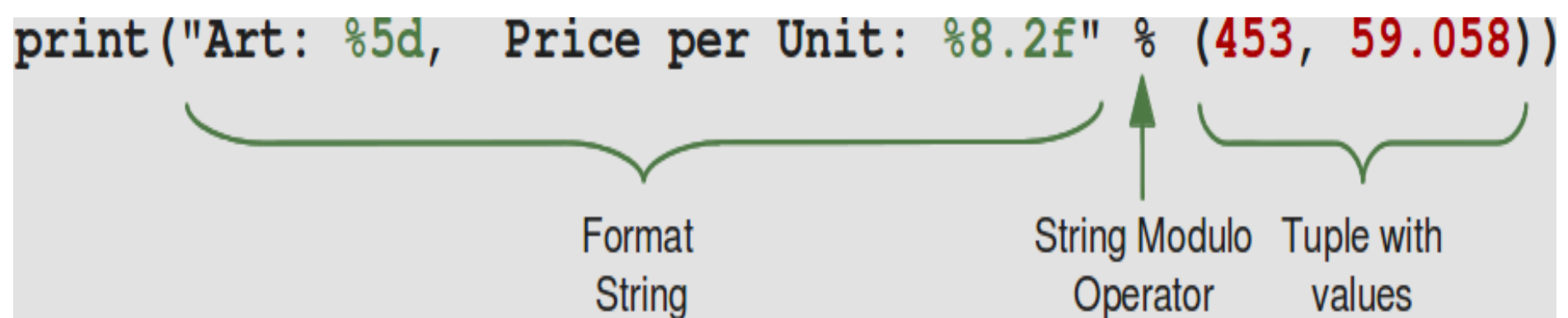
- `a , b , c = 'python', 9 , 7.1345672`
- **# Unformatted printing**
- `print(a,b,c)`
- ***# Formatted printing***
- `print(' %s%d%f ' % (a,b,c))`
- ***# Old style printing align right***
- `print(' %10s%8d%4.2f ' % (a,b,c))`

# Padding and aligning using %

a,b= 453, 59.058

print("Art: %5d, Price per unit: %8.2f" % (a,b))

print("Art: %5d, Price per unit: %8.2f" % (453, 59.058))



The diagram shows the print statement with three green curly braces and an arrow pointing to the modulo operator. The first brace is under "Art: %5d, Price per Unit: %8.2f" and is labeled "Format String". The second brace is under the modulo operator "%" and is labeled "String Modulo Operator". The third brace is under "(453, 59.058)" and is labeled "Tuple with values".

```
print("Art: %5d, Price per Unit: %8.2f" % (453, 59.058))
```

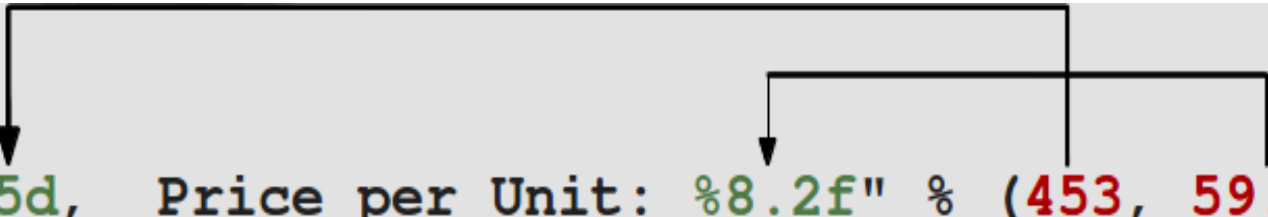
Format String

String Modulo Operator

Tuple with values



```
print("Art: %5d, Price per Unit: %8.2f" % (453, 59.058))
```



output



String Modulo Operator

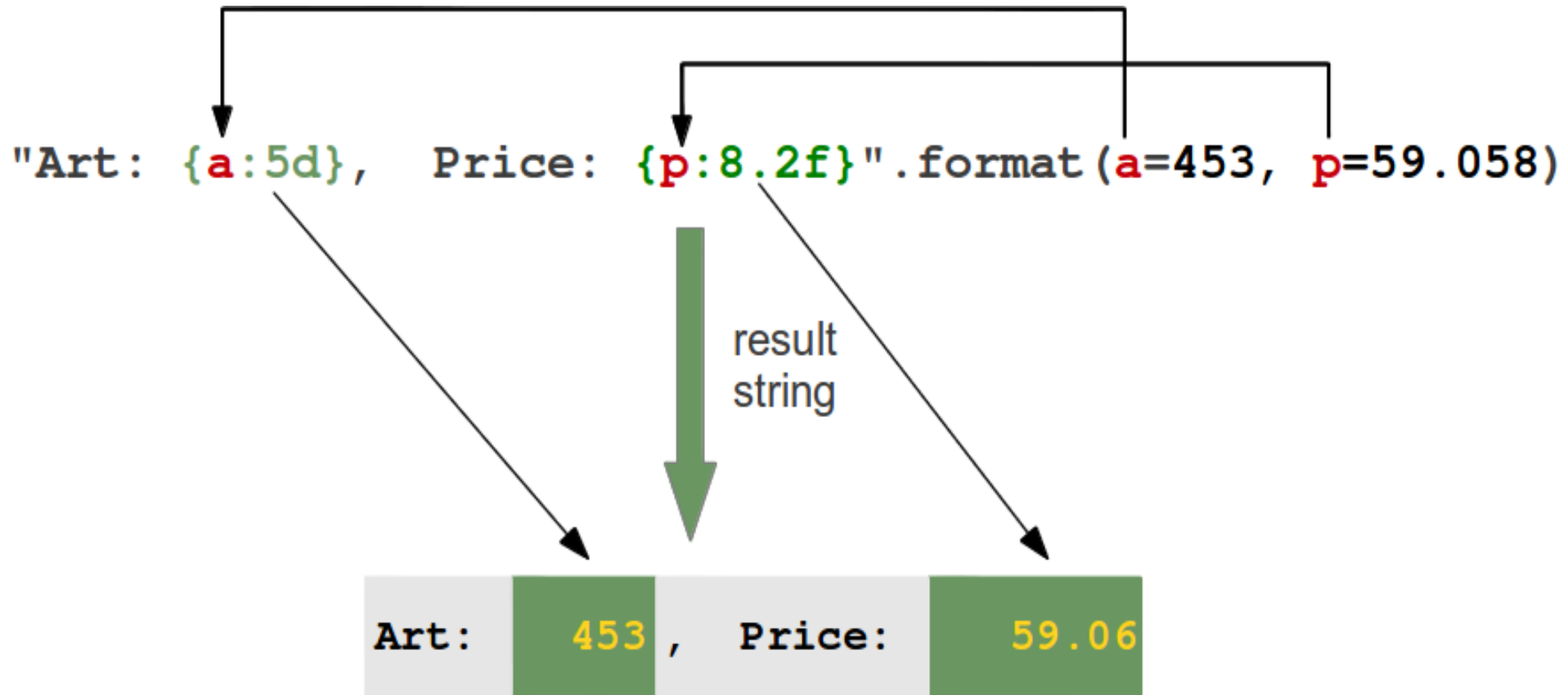
```
Art: 453, Price per Unit: 59.06
```

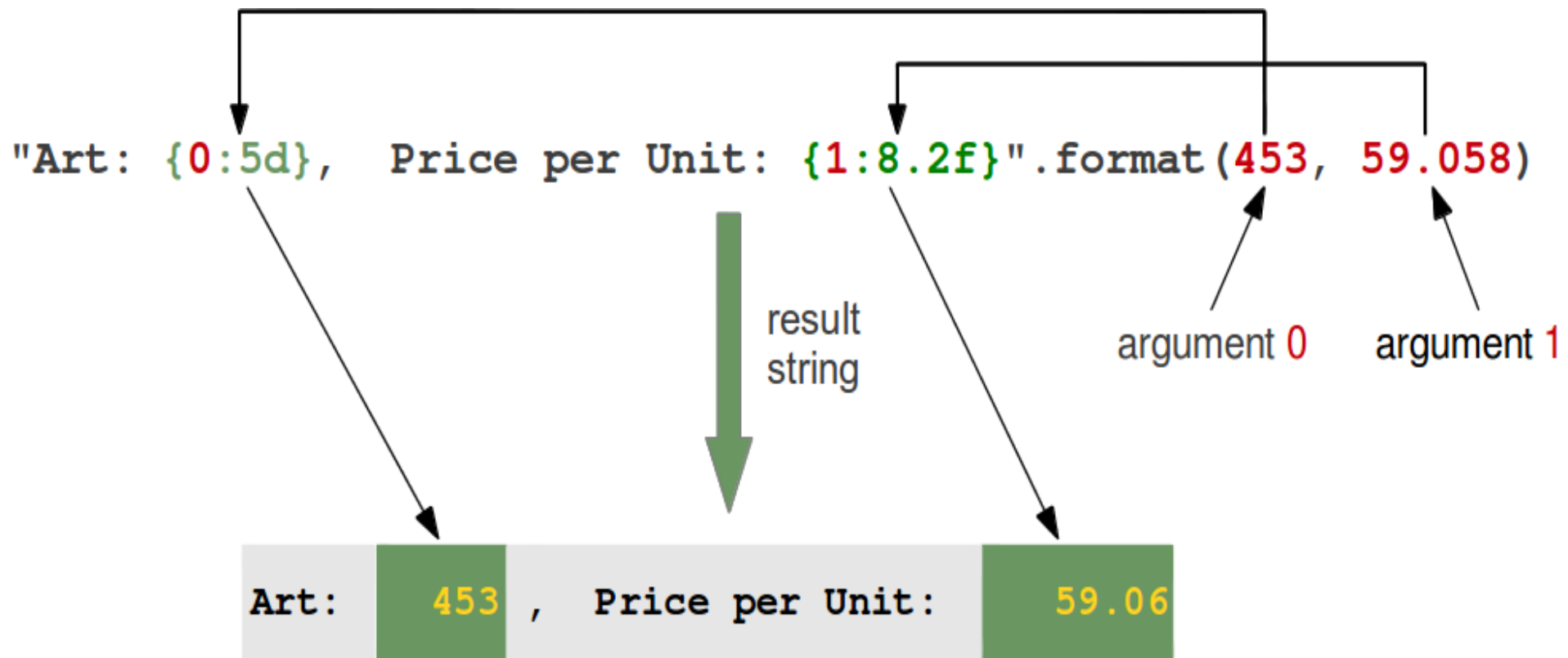
Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

- `a=79`
- `print("%e"%a)`      `# 7.900000e+01`
- `print("%o"%a)`      `# 117`
- `print("%x"%a)`      `# 4f`
- `print("%X"%a)`      `# 4F`
- `print("%c"%a)`      `# H`

# Padding and aligning using **.format()**

- `a , b , c ='python', 9 ,7.1345672`
- ***#New formatted printing***
- `print('{} {} {}'.format(a,b,c))`
- ***#New formatted printing align right***
- `print('{:>10} {:8d} {:4.2f}'.format(a,b,c))`
- **> right aligned**(since **by default left aligned**)
- **:8d** left aligned 8 digit integer





- **Align right:**
- `print('%10s' % ('test',))`      **#Old style**
- `print('{:>10}'.format('test'))` **#New style**
- `print("%010d"%(a))`      **# 00000000065**
- `print('%-10s' % ('test',) )`      **#Align left Old style**
- `print('{:10}'.format('test') )` **# Align left New style**
- `Print('{:^10}'.format('test'))` **#Align center New**
- `'{:*<15}'.format('Python')` **#Python\*\*\*\*\***
- **Truncating long strings**
- `Print('% .5s' % ('xylophone',) )`      **#Old style**
- `Print('{: .5}'.format('xylophone'))`      **#New style**

# Combining truncating and padding

- `print('%-10.5s' % ('xylophone',))` **#Old style**
- `print('{:10.5}'.format('xylophone'))` **#New style**
- **Padding numbers:**
- Similar to strings numbers can also be constrained to a specific width.
- `print(' %4d ' % (42) )` **#Old style**
- `print('{:4d}'.format(42))` **#New style**
- `a=65`
- `print("%+d"% (a))` *# prints +65*

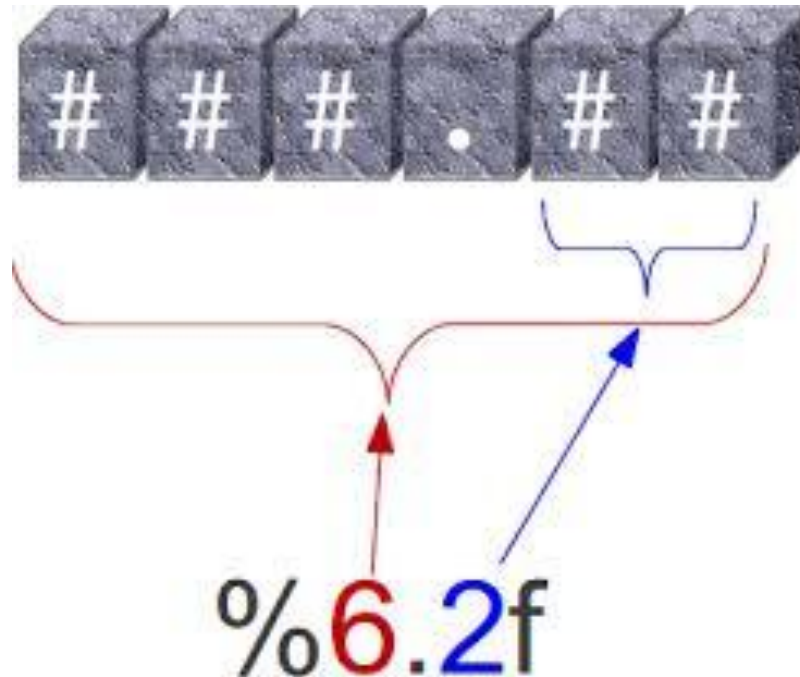


- This option signals the use of a comma for a thousands separator.
- `x=78962324245`
- `print("The value is {:,}".format(x))`
- The value is 78,962,324,245

# Padding and Truncating

## Floating point numbers

- `print('%6.2f' % (3.141592653589793,))` **#Old style**
- `print('{:6.2f}'.format(3.141592653589793))` **#New**
- **Output:** 3.14



# Precision for floating point numbers

- Again similar to truncating strings the precision for floating point numbers limits the number of positions after the decimal point.
- For floating points the **padding value represents the length of the complete output.**
- If we want our output to have at least 6 characters with 2 after the decimal point.
- `print('%06.2f' % (3.141592653589793,))` **#Old style**
- `print('{:06.2f}'.format(3.141592653589793))` **#New style**
- **Output**
- **003.14**

# Getitem and Getattr

- New style formatting allows even greater flexibility in accessing nested data structures.
- It supports accessing containers that support `__getitem__` like for example dictionaries and lists:
- `person = {'first': 'Jean-Luc', 'last': 'Picard'}`
- `Print('{p[first]} {p[last]}'.format(p=person))`
- **Output**
- Jean-Luc Picard
- -----
- `data = [4, 8, 15, 16, 23, 42]`
- `print('{d[4]} {d[5]}'.format(d=data))`
- **Output**
- 23 42

# Execute a Python Script

- Execution of python program means **execution of the byte code on Python Virtual Machine(PVM)**
- If Python has write-access for the directory where the Python program resides, it will **store the compiled byte code in a file that ends with a .pyc suffix.**
- This file has to be newer than the file with the .py suffix. If such a file exists, Python will load the byte code, which will speed up the start up time of the script.
- **If there exists no byte code version, Python will create the byte code before it starts the execution of the program.**

- Execution of a Python program means execution of the byte code on the **Python Virtual Machine (PVM)**.
- If Python has **no write access**, the program will work anyway. The **byte code will be produced but discarded when the program exits**.  
Whenever a Python program is called, Python will check, if there exists a compiled version with the .pyc suffix.

## Execute a Python Script

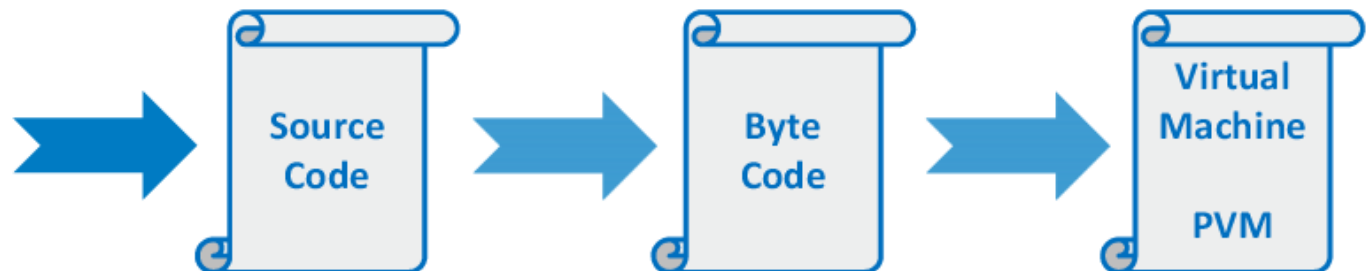
- Execution of python program means execution of the byte code on Python Virtual Machine

```
print("Hello World!")
```

#prints Hello World!

```
print("My first sample python script")
```

#prints My first sample python script



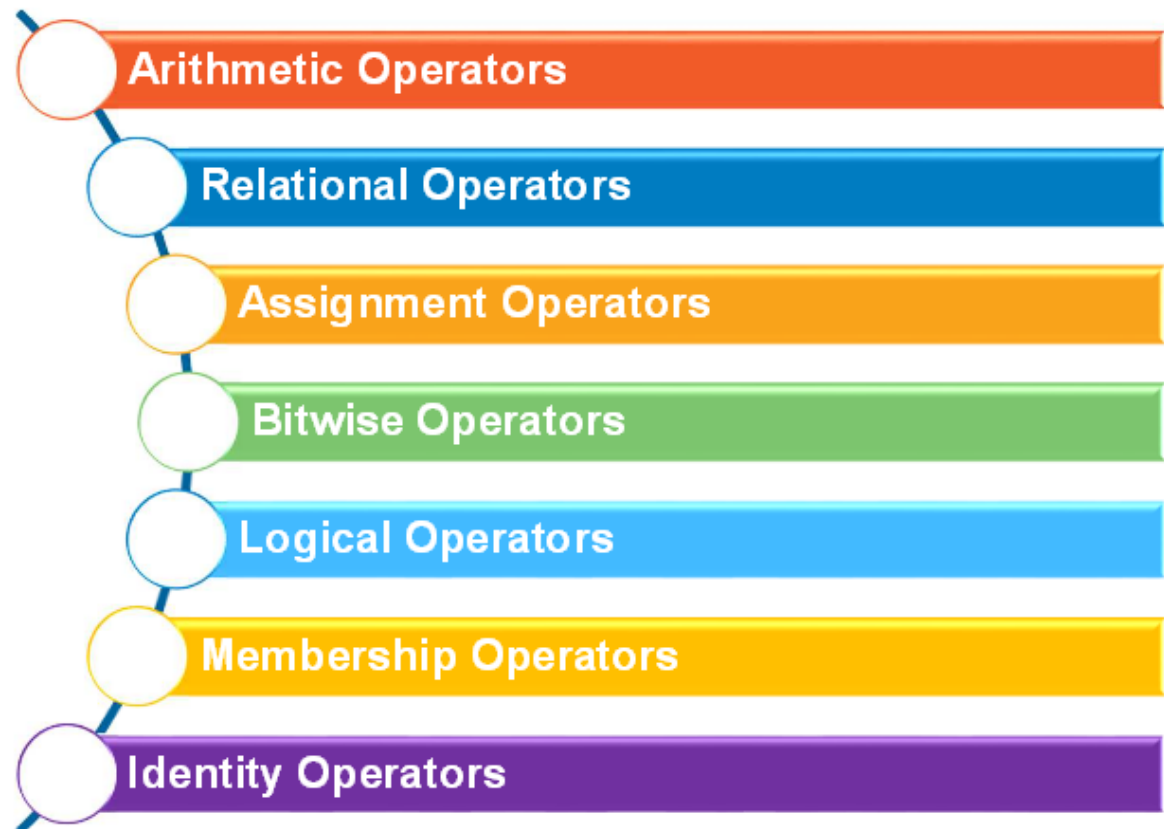
# Method for taking input from the console

- We use `input()` function which takes input as string
- `st=input("Enter your name")` *#takes input as string*
- `age=int(input("Enter your age"))` *#takes input as int*
- `sal=input("Enter your salary")`
- `sal=sal+3000` *# will give error because sal is string,*  
*can not be added to integer 3000*
- `x=eval(input("enter expression"))` *# will evaluate*  
*the input expression and value is assigned to x*
- If the input given is `5+3-6` which is an expression
- `print(x)` *#will print 2*



# Operators (1 of 7)

- Used to perform specific operations on one or more operands (or variables) and provide a result



## Operators (2 of 7)

- **Arithmetic Operators**
  - Used for performing arithmetic operations

Operators	Description	Example
+	Additive operator (also used for String concatenation)	$2 + 3 = 5$
-	Subtraction operator	$5 - 3 = 2$
*	Multiplication operator	$5 * 3 = 15$
/	Division operator	$6 / 2 = 3$
%	Modulus operator	$7 \% 2 = 1$
//	Truncation division (also known as floor division)	$10 // 3 = 3$ $10.0 // 3 = 3.0$
**	Exponentiation	$10 ** 3 = 1000$

# % operator

- `print(8 % 3)` #  $8 = 2 * 3 + 2$
- Output: 2
- `print(-8 % 3)` #  $-8 = -3 * 3 + 1$
- Output: 1, because remainder is always positive
- `print(-7 % 3)` #  $-7 = -3 * 3 + 2$
- Output: 2
- `print(2.8 % 1.3)`
- Output: 0.20

// operator also called **floor division** operator

- `print(5/2)`      **# 2.5**
- `print(5//2)`    **# 2**
- `print(5.0//2)` **# 2.0**

## Operators (3 of 7)

- **Relational Operators**
  - Also known as **Comparison operators**
  - Used in conditional statements to compare values and take action depending on the result

Operators	Description
==	Equal to
<	Less than
>	Greater than
<=	Lesser than or equal to
>=	Greater than or equal to
!=	Not equal to
<>	Similar to Not equal to

# Assignment Operators

Operator	Description	Example
<b>=</b>	Assigns values from right side operands to left side operand	<code>c = a + b</code> assigns value of <code>a + b</code> into <code>c</code>
<b>+=</b>	It adds right operand to the left operand and assign the result to left operand	<code>c += a</code> is equivalent to <code>c = c + a</code>
<b>-=</b>	It subtracts right operand from the left operand and assign the result to left operand	<code>c -= a</code> is equivalent to <code>c = c - a</code>
<b>*=</b>	It multiplies right operand with the left operand and assign the result to left operand	<code>c *= a</code> is equivalent to <code>c = c * a</code>
<b>/=</b>	It divides left operand with the right operand and assign the result to left operand	<code>c /= a</code> is equivalent to <code>c = c / a</code>
<b>%=</b>	It takes modulus using two operands and assign the result to left operand	<code>c %= a</code> is equivalent to <code>c = c % a</code>
<b>**=</b>	Performs exponential calculation on operators and assign value to the left operand	<code>c **= a</code> is equivalent to <code>c = c ** a</code>
<b>//=</b>	It performs floor division on operators and assign value to the left operand	<code>c //= a</code> is equivalent to <code>c = c // a</code>

# Multiple Assignments

- **Multiple Assignments:**
- If students Ram, Shyam and John belong to semester 6 then values can be assigned as
- Ram = Sham = John = 6
- a, b, c = 10, 20, 30 **is same as**
- a = 10, b = 20, c = 30
- x, y, z="John", 2, 3.5 **is same as**
- X="John", y=2, z=3.5
- a,b=b,a **#will swap the references and values pointed a and b will be swapped**

# Bitwise operators

Operator	Description	Example
	Let <b>a = 60 = 0011 1100</b>	<b>b = 13 = 0000 1101</b>
<b>&amp;</b> Binary AND	Operator copies a bit to the result if it exists in both operands	(a <b>&amp;</b> b) = 12 (means <b>0000 1100</b> )
<b> </b> Binary OR	It copies a bit if it exists in either operand.	(a <b> </b> b) = 61 (means <b>0011 1101</b> )
<b>^</b> Binary XOR	It copies the bit if it is set in one operand but not both.	(a <b>^</b> b) = 49 (means <b>0011 0001</b> )



# Bitwise operators

- `a=60`
- `b=13`
- `print(a & b)` **# 12**
- `print(a | b)` **# 61**
- `print(a ^ b)` **# 49**
- **0011 1100 & 0000 1101 = 0000 1100**
- **0011 1100 | 0000 1101 = 0011 1101**
- **0011 1100 ^ 0000 1101 = 0011 0001**

# Bitwise operators

<b>~</b> Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	$(\sim a) = -61$ (means <b>1100 0011</b> in <b>2's complement</b> form due to a signed binary number.
<b>&lt;&lt;</b> Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	$a \ll 2 = 240$ (means 1111 0000) (multiplied by 4)
<b>&gt;&gt;</b> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	$a \gg 2 = 15$ (means 0000 1111) (divided by 4)

- `a = 60`
- `print(~a)`      **# -61, 2's complement**
- `print(a << 2)`   **# 240 ,multiplying by 4**
- `print(a >> 2)`   **# 15, dividing by 4**
- `print(a >> 3)`   **# 7, dividing by 8**

# Logical Operators

- Are based on Boolean Algebra –
- Returns result as either True or False

Operator	Meaning
and	Short Circuit-AND
or	Short Circuit-OR
not	Unary NOT

- `print(bool('hello'))`
- **# True, since python treats boolean value of all strings as “True”**
- `print(not "hello")` **# False**
- `print(not "")`
- **# True, python treats boolean value of an empty string as “False”**

# Membership Operators

Test for **membership in a sequence**, such as strings, lists, or tuples.

Operator	Meaning
<b>in</b>	Returns <b>true</b> if given <b>element is found</b> in given sequence else false
<b>not in</b>	Returns <b>true</b> if given <b>element is not found</b> in given sequence else false

- `s='psit'`
- `x='i' in s`
- `print(x)` #print true since 'i' is **in** s
- `y='i' not in s`
- `print(y)` # print false since 'i' is in s
- `z='f' not in s`
- `print(z)` # print true since 'f' is **not in** s

# Identity Operators

Are used to **compare memory locations** or **addresses** of 2 objects

Operator	Meaning
<b>is</b>	Returns true if variables or objects on both sides of operator are referring to same object else false
<b>is not</b>	Returns false if variables or objects on both side of operator are referring to same object else true

- `a,b = 20, 30`
- `print(id(a),id(b))` # will print **different memory adderssese**
- `print(a is b)` # `id(a) == id(b)` will print **false**, a and b points to different area

# Identity operator

- `a = 20`
- `b = 30`
- `print(id(a),id(b))` # will print different memory addersses
- `print(a is b)` # print false, points to different location
- `a=b` # a and b now point to the integer object 30
- `print(a is b)` # print true, points to same location
- `print(a is not b)` # print false, id of a and b are same



# Operator Precedence

- Operator precedence affects how an expression is evaluated.
- For example,  $x = 7 + 3 * 2$ ; here,  $x$  is assigned 13, not 20 because operator  $*$  has higher precedence than  $+$ , so it first multiplies  $3*2$  and then adds into 7.
- Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.
- When arithmetic operators have **same precedence**, they are **left associative**. E.g.  $a-b-c$  is equivalent to  $(a-b)-c$
- Whereas **exponentiation operator**  $**$  is the **right associative**.
- We can **use parentheses to override the rules**,
- so we can write  $a-(b-c)$

Operator	Description
<b>**</b>	<b>Exponentiation (raise to the power)</b>
<b>~ ,+, -</b>	<b>Complement, unary plus and minus (method names for the last two are +@ and -@)</b>
<b>*, /, %, //</b>	<b>Multiply, divide, modulo and floor division</b>
<b>+, -</b>	<b>Addition and subtraction</b>
<b>&gt;&gt;, &lt;&lt;</b>	<b>Right and left bitwise shift</b>
<b>&amp;</b>	<b>Bitwise 'AND'</b>
<b>^,  </b>	<b>Bitwise exclusive `OR' and regular `OR'</b>
<b>&lt;= , &lt;, &gt;, &gt;=</b>	<b>Comparison operators</b>
<b>&lt;&gt;, ==, !=</b>	<b>Equality operators</b>
<b>=, %=, /=, //=, -= +=, *=, **=</b>	<b>Assignment operators</b>
<b>is, is not</b>	<b>Identity operators</b>
<b>in, not in</b>	<b>Membership operators</b>
<b>not, or, and</b>	<b>Logical operators</b>

# Control structures

- Decision making statements
  1. if-else
  2. if-elif-else
- **Note:**
- There is no switch case statement in Python
- We can do this easily enough
  1. with a sequence of **if... elif... elif...else**“
  2. And that you can use dictionary mapping for functions and dispatch methods for classes.

# Replacement of switch case

- **def** numbers\_to\_strings(argument):
- switcher = {
- 0: "zero",
- 1: "one",
- 2: "two",
- }
- **return** switcher.get(argument, "nothing")
- Which is same as given in next slide in 'C' code

- **function(argument){**
- **switch(argument) {**
- **case 0:**
- **return "zero";**
- **case 1:**
- **return "one";**
- **case 2:**
- **return "two";**
- **default:**
- **return "nothing";**
- **};**

# Iterative Statements–

- Loop statements:
- Allows repeated execution of a statement or a set of statements multiple times based on the specified condition or range.
  1. While Loop
  2. For Loop
  3. Range
- Loop Control Statements:
- Are used to **change flow of execution** from its normal sequence-
  - 1. Break
  - 2. Continue
  - 3. Pass

# Indentation in Python

- Python uses **offside rule** notation for coding
- ***Uses indentation for blocks, instead of curly brackets***
- ***The delimiter followed in Python is a colon (:) and indented spaces or tabs***

# if-else statement

- `x= 3`
- `if x > 5:`      **#Press enter after colon(:)**  
    `print("true")`  
    `print(x)`
- `else:`  
    `print ("false")`  
    `print("still in else block")`
- `print ("Out of if block")`



# if-elif-else statement syntax

- if condition1:  
    statement(s)
- elif condition2:  
    statement(s)
- elif condition3:  
    statement(s)
- else:  
    statement(s)

# if-elif-else

- `a,b,c = 23,74,90`
- `if a>b and a>c:`
- `print("largest value is:")`
- `print(a)`
- `elif b>a and b>c:`
- `print("largest value is:")`
- `print(b)`
- `else:`
- `print("largest value is:")`
- `print(c)`

# while loop

- Repeats execution of a statement or a set of statements while a given condition is TRUE.
- Checks the condition each time before executing the statements in body of loop.
- `x=1`
- `while x<=7:`
  - `print(x)`
  - `x+=1`

# for loop

- Repeats execution of a sequence of statements for a specific number of times
- **Syntax:**
- **for** variable **in** sequence:  
    statement\_1  
    statement\_2  
    .  
    statement\_n

- **for x in 1,2,'hello', 7,'world',5.16:  
    **print(x)****

**Out put:**

**1**

**2**

**hello**

**7**

**world**

**5.16**

- **for y in “python”:** **# iterating over string**  
**print(y)**

**Out put:**

**p**

**y**

**t**

**h**

**o**

**n**

# range( ) function

- Range is a built-in function that creates a list of integers.
- **range(6)**                      *# creates a list of integers*
- **range(1,6)**                      *# 1, 2, 3, 4, 5*
- **range(0,6,2)**                      *# 0, 2, 4, increments of 2*
- **range(6,1,-2)**                      *# 6,4,2, decrements of 2*

# range( ) function in loops

- Used in case the need is to iterate over a specific number of times within a given range in steps/intervals mentioned. Syntax is :
- **range(start, stop, step)**
- **step=increment(+)/decrement(-)**
- **start (inclusive)**
- **stop (exclusive)**
- Return an object that produces a sequence of integers from **start (inclusive)** to **stop (exclusive)** by step.



# range() function in loops

**range(lower limit\*, upper limit, Increment\_by/decrement\_by\*)**

**\* optional**

Loop	Out put	Explanation
for i in <b>range(6)</b> : print(i)	0,1,2,3,4,5	Prints all the values in given range from 0, <b>exclusive of upper limit</b>
for i in <b>range(1,6)</b> : print(i)	1,2,3,4,5	Prints all the values in given range exclusive of upper limit
for i in <b>range(0,6,2)</b> : print(i)	0,2,4	Prints values in given range in <b>increments of 2</b>
for i in <b>range(6,1,-2)</b> : print(i)	6,4,2	Prints values in given range in <b>decrements of 2</b>
for ch in <b>"Hello World"</b> : print(ch)		Prints all the characters in the string

# for-else

- for i in **range(1,6):**
- print(i)
- else:
- print(**“loop completed successfully”**)
- -----
- for i in **range(1,6):**
- print(i)
- if i==3:
- print(**"loop not completed successfully"**)
- break
- else:
- print(**"loop completed successfully"**)

# break

- When an **external condition is triggered**, **Exits a loop** immediately.
- Break Statement: Terminates execution of loop statements and resumes execution at statement immediately following the loop. e.g.
- `x=1`
- `while x<=7:`
- `print(x)`
- `x+=1`
- `if x==5:`
- `break`
- `print("out of loop")`

# continue

- Causes the next iteration of the loop to execute and immediately retest its condition prior to reiterating.
- `x=1`
- `while x<=7:`
- `if x==5:`
- `x=x+1`
- `continue`
- `print(x)`
- `x=x+1`
- `print("out of loop")`

# pass

- Pass statement is never executed.
- Used when a **statement is syntactically required** and do not want any code to execute now.
- If there is a need to implement code in future.
- Behaves like a **placeholder for future code.**

# Immutability

- A data type is immutable if it is not possible to change the value of an object of that type.
- The following are **immutable** and **mutable** objects:

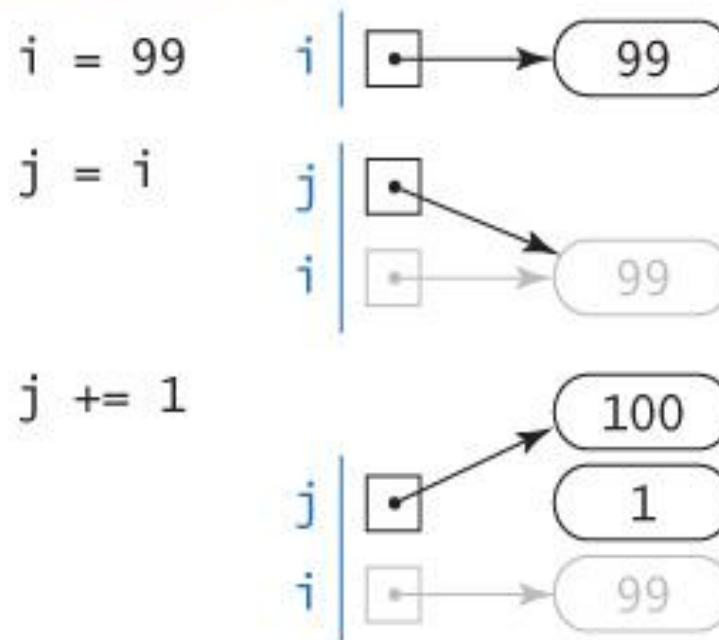
Immutable	Mutable
Numeric type: int, float, complex	List
Bool: bool	Dictionary
String: str	Set
Tuple	byte array
Frozen set	

- In an immutable data type, operations that might seem to **change a value actually result in the creation of a new object.**

*informal trace*

	i	j
i = 99	99	
j = i	99	99
j += 1	99	100

*object-level trace*



*Immutability of integers*

# aliasing

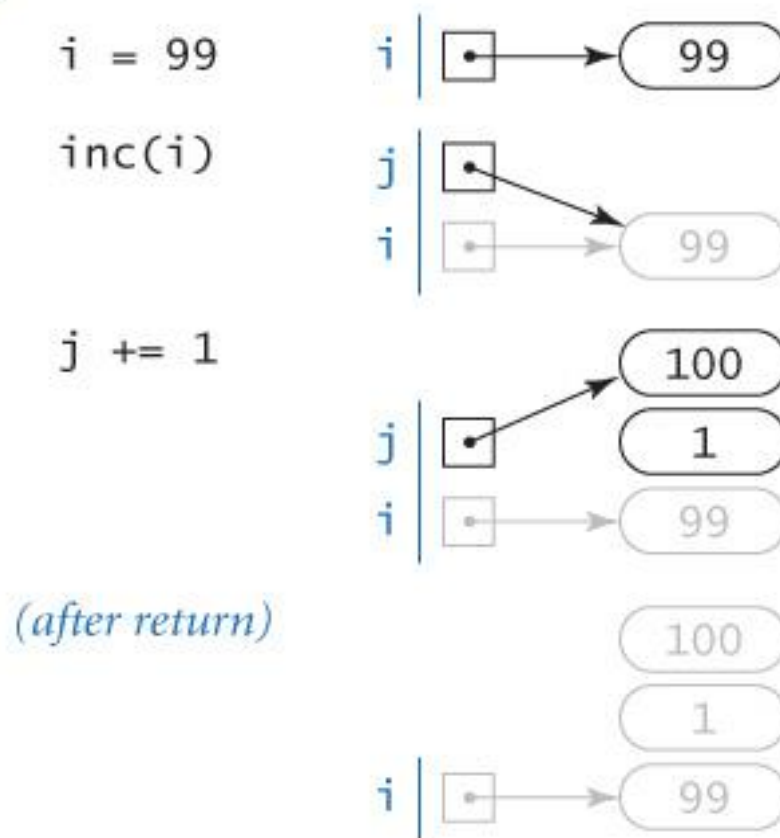
- When we pass arguments to a function, the **arguments and the function's parameter variables become aliases.**
- **i=99**
- **def inc(j) :**
- **j=j+1**



### *informal trace*

	<u>i</u>	<u>j</u>
i = 99	99	
inc(i)	99	99
j += 1	99	100
(after return)	99	100

### *object-level trace*



*Aliasing in a function call*

# WHEN MUTABILITY MATTERS

- Mutability might seem like an innocuous topic, but when writing an efficient program it is essential to understand. For instance, the following code is a straightforward solution to concatenate a string together:
  - `string_build = "";`
  - `for data in container:`
  - `string_build += str(data)`
  - `print(string_build)`
- But in reality, it is extremely inefficient. Because strings are immutable, concatenating two strings together actually creates a third string which is a combination of the previous two.

- If we are iterating a lot and building a large string, you will waste a lot of memory creating and throwing away objects.
- Also, do not forget that nearing the **end of the iteration** we will be **allocating and throwing away very large string objects** which is even more **costly**.
- The following is a **more efficient and pythonic** method:
  - **builder\_list = []**
  - **for data in container:**
    - **builder\_list.append(str(data))**
  - **"".join(builder\_list)**

- Better way is to **use a generator**
- Which is **cleaner code and runs faster**
- **"".join(str(data) for data in container)**
- This code takes advantage of the mutability of a single list object to gather your data together and then allocate a single result string to put your data in.
- That cuts down on the total number of objects allocated by almost half.

# Recursion

- Recursive programming is directly related to mathematical induction, a technique for proving facts about discrete functions.