A Quick Tour of Python

TRILOCHAN OJHA

Dept. of LIVEWIRE

Acknowledgements

- MS Office clip art, various websites and images
  - The images/contents are used for teaching purpose and for fun. The copyright remains with the original creator. If you suspect a copyright violation, bring it to my notice and I will remove that image/content.

Write/Edit

Run
with some input

OK?

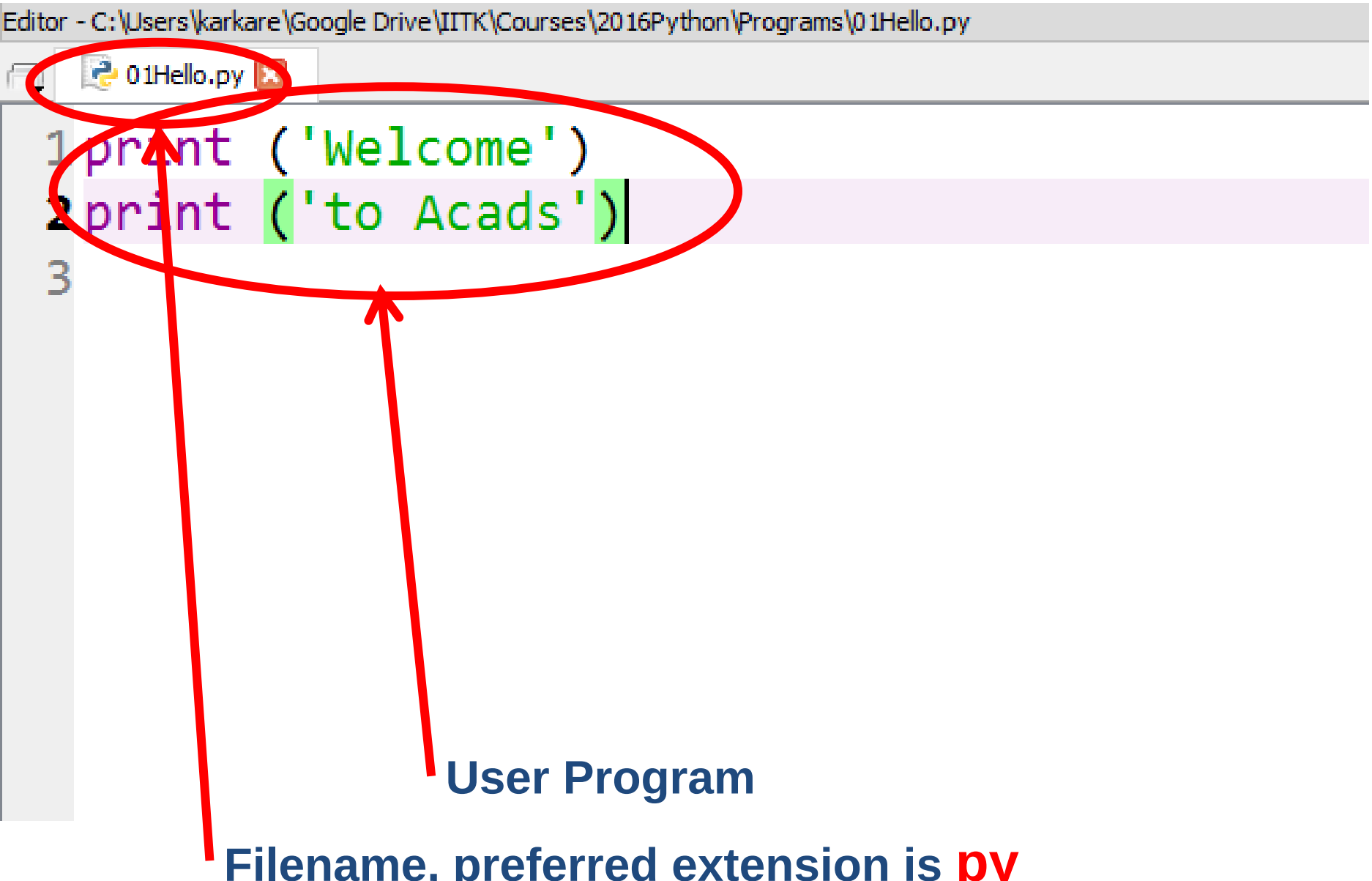
NO

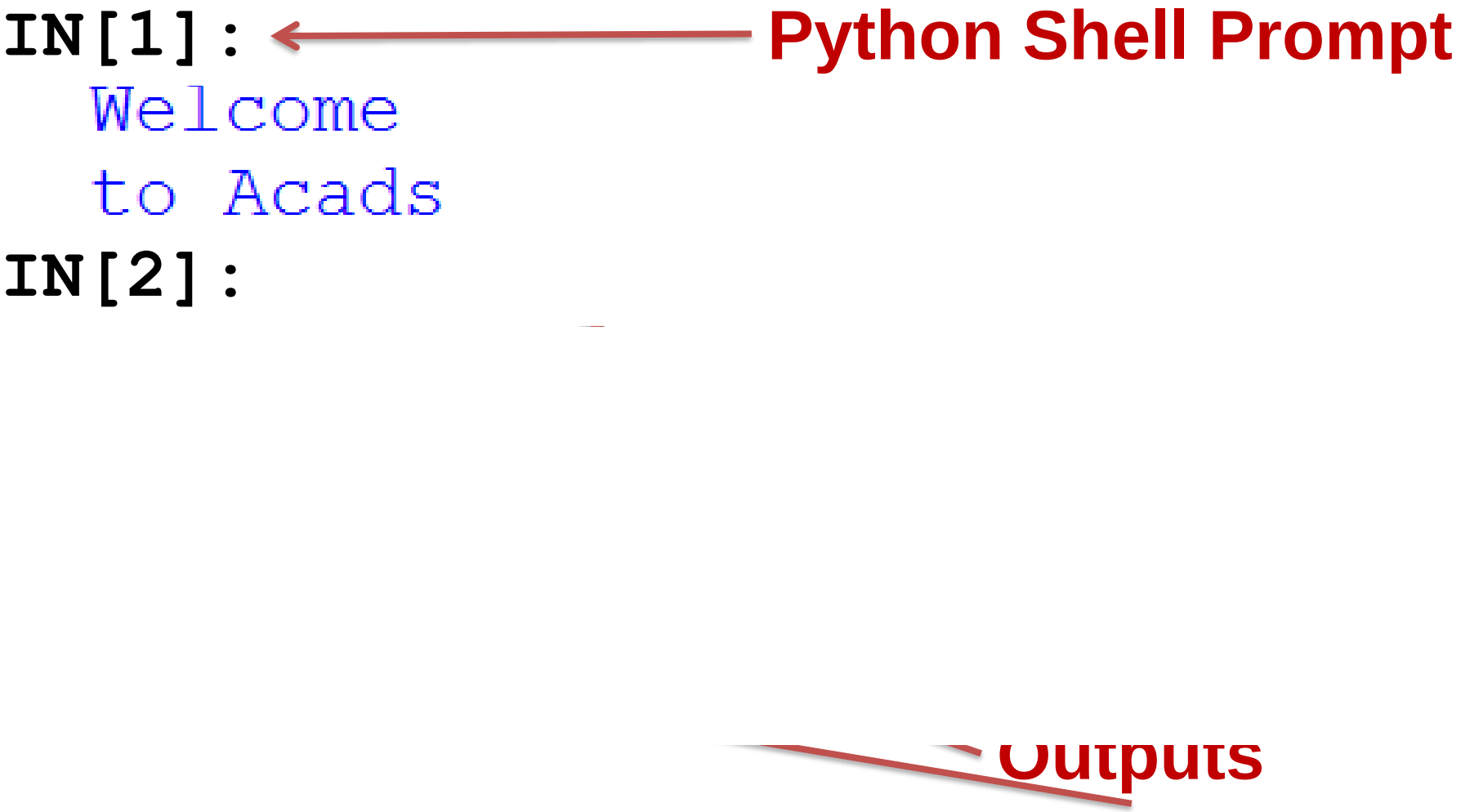
YES

More Inputs?

YES

NO


Editor - C:\Users\karkare\Google Drive\IITK\Courses\2016Python\Programs\01Hello.py
01Hello.py
1 print ('Welcome')
2 print ('to Acads')
3
User Program
Filename, preferred extension is py



Python Shell is Interactive

Interacting with Python Programs

- Python program communicates its results to user using print
- Most useful programs require information from users
  - Name and age for a travel reservation system
- Python 3 uses input to read user input as a string (str)

input

- Take as argument a string to print as a prompt
- Returns the user typed value as a string
  - details of how to process user string later

```
IN[1]: age =      input('How old are you?')

IN[2]:

IN[3]:
```

Elements of Python

- A Python program is a sequence of **definitions** and **commands (statements)**
- Commands manipulate **objects**
- Each object is associated with a **Type**
- **Type:**
  - A set of values
  - A set of operations on these values
- **Expressions**: An operation (combination of objects and **operators**)

Types in Python

- **int**
  - Bounded integers, e.g. 732 or -5
- **float**
  - Real numbers, e.g. 3.14 or 2.0
- **long**
  - Long integers with unlimited precision
- **str**
  - Strings, e.g. ‘hello’ or ‘C’

Types in Python

- **Scalar**
  - Indivisible objects that do not have internal structure
  - **int** (signed integers), **float** (floating point), **bool** (Boolean), ***NoneType***
    - NoneType is a special type with a single value
    - The value is called **None**
- **Non-Scalar**
  - Objects having internal structure
  - **str** (strings)

Example of Types

```
In [14]: type(500)
Out[14]: int
```

Type Conversion (Type Cast)

- Conversion of value of one type to other
- We are used to int ↔ float conversion in Math
  - Integer 3 is treated as float 3.0 when a real number is expected
  - Float 3.6 is truncated as 3, or rounded off as 4 for integer contexts
- Type names are used as type converter functions

Type Conversion Examples

```
In [20]: int(2.5)
Out[20]: 2

In [21]: int(2.3)
Out[21]: 2

In [22]: int(3.9)
Out[22]: 3

In [23]: float(3)
Out[23]: 3.0

In [24]: int('73')
Out[24]: 73

In [25]: int('Acads')
Traceback (most recent call last):

  File "<ipython-input-25-90ec37205222>", line 1, in <module>
    int('Acads')

ValueError: invalid literal for int() with base 10: 'Acads'
```

Note that float to int conversion is truncation, not rounding off

```
In [26]: str(3.14)
Out[26]: '3.14'

In [27]: str(26000)
Out[27]: '26000'
```

Type Conversion and Input

```
In [11]: age = input('How old are you? ')

How old are you? 35

In [12]: print ('In 5 years, your age will be', age + 5)
```

```
In [13]: print ('In 5 years, your age will be', int(age) + 5)
In 5 years, your age will be 40
```

Operators

- Arithmetic

+	-	*	//	/	%	**

- Comparison

==	!=	>	<	>=	<=

- Assignment

=	+=	-=	*=	//=	/=	%=	**=

- Logical

and	or	not

- Bitwise

&	|	^	~	>>	<<

- Membership

in	not in

- Identity

is	is not

Variables

- A name associated with an object
- Assignment used for binding

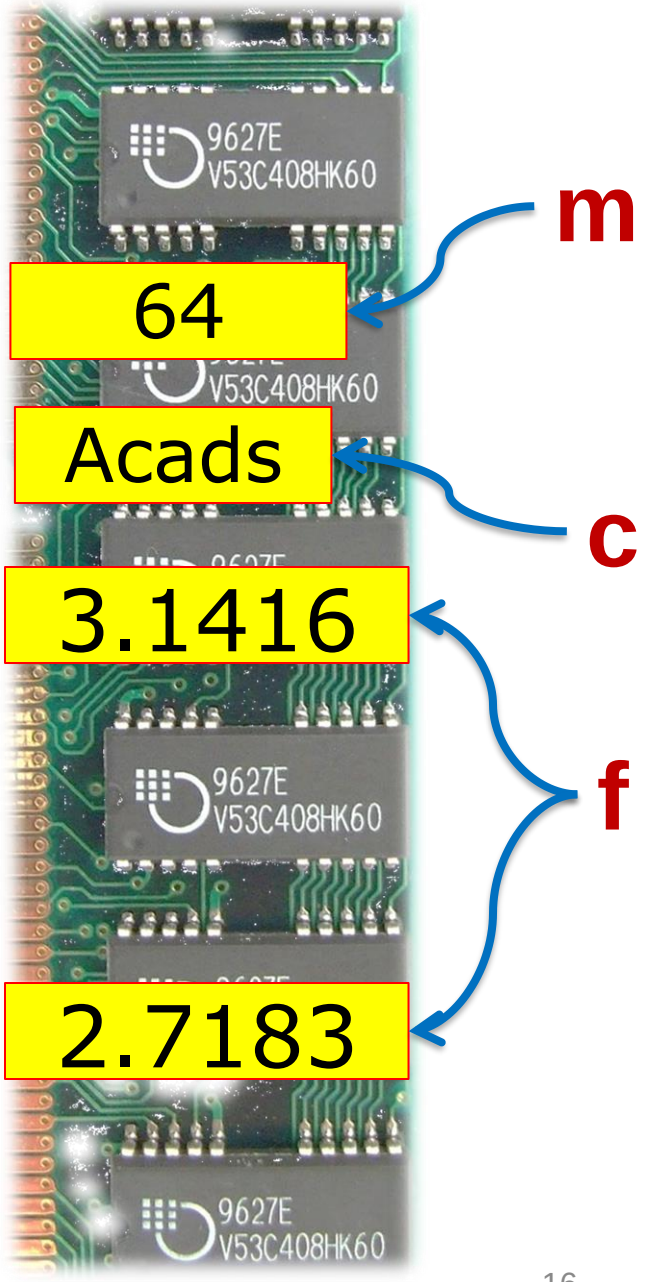
m = 64;

c = ‘Acads’;

f = 3.1416;

- Variables can change their bindings

f = 2.7183;



Assignment Statement

- A simple assignment statement

Variable = Expression;

- Computes the value (object) of the expression on the right hand side expression (RHS)
- Associates the name (variable) on the left hand side (LHS) with the RHS value
- = is known as the assignment operator.

Multiple Assignments

- Python allows multiple assignments

```
x, y = 10, 20
```

Binds x to 10 and y to 20

- Evaluation of multiple assignment statement:
  - All the expressions on the RHS of the = are first evaluated **before any binding happens**.
  - Values of the expressions are bound to the corresponding variable on the LHS.

```
x, y = 10, 20
x, y = y+1, x+1
```

x is bound to 21 and y to 11 at the end of the program

Programming using Python

Operators and Expressions

Binary Operations

Op	Meaning	Example	Remarks
+	Addition	9+2 is 11	
		9.1+2.0 is 11.1	
-	Subtraction	9-2 is 7	
		9.1-2.0 is 7.1	
*	Multiplication	9*2 is 18	
		9.1*2.0 is 18.2	
/	Division	9/2 is 4.25	In Python3
		9.1/2.0 is 4.55	Real div.
//	Integer Division	9//2 is 4	
%	Remainder	9%2 is 1	

The // operator

- Also referred to as “integer division”
- Result is a whole integer (floor of real division)
  - But the type need not be int
  - the integral part of the real division
  - rounded towards minus infinity ($-\infty$)
- Examples

9//4 is 2	**(-1)//2 is -1**	**(-1)//(-2) is 0**
1//2 is 0	**1//(-2) is -1**	**9//4.5 is 2.0**

The % operator

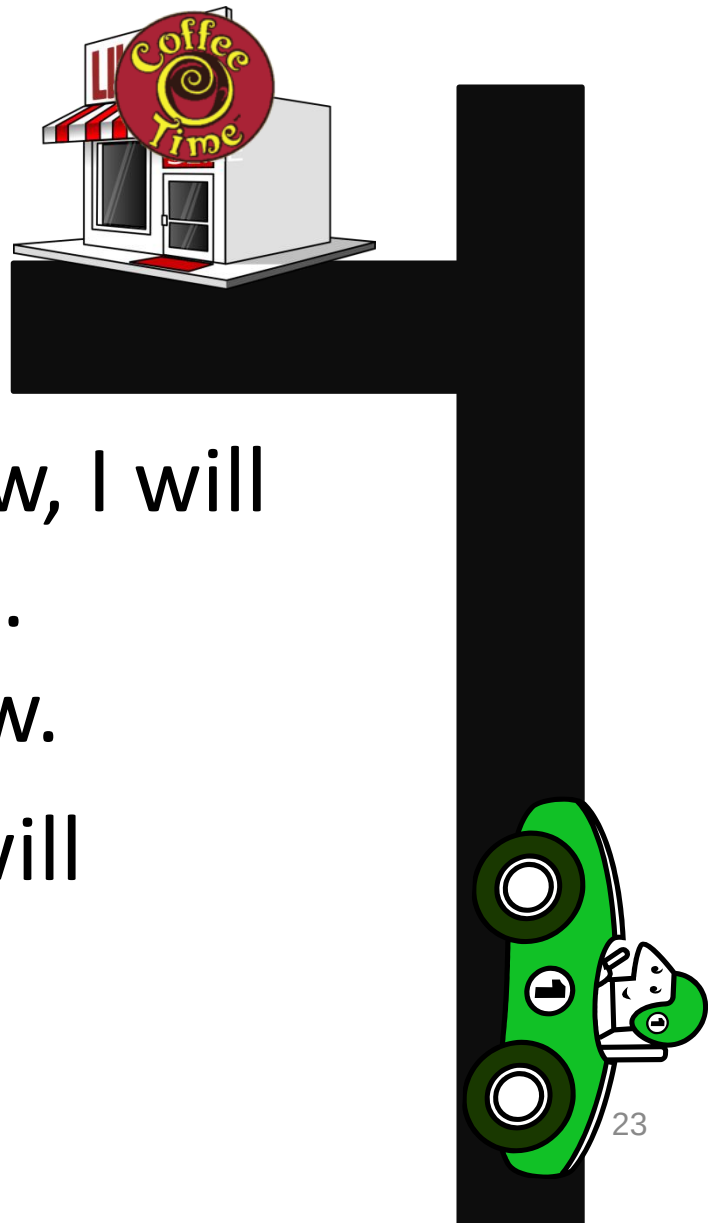
- The remainder operator % returns the remainder of the result of dividing its first operand by its second.

9%4 is 1	**(-1)%2 is 1**	**(-1)//(-2) is 0**
9%4.5 is 0.0	**1%(-2) is 1**	**1%0.6 is 0.4**

Ideally: x == (x//y)*y + x %y

Conditional Statements

- In daily routine
  - If it is very hot, I will skip exercise.
  - If there is a quiz tomorrow, I will first study and then sleep. Otherwise I will sleep now.
  - If I have to buy coffee, I will go left. Else I will go straight.



if-else statement

- Compare two integers and print the min.

```
if  x < y:
        print (x)
else:
        print (y)
print ('is the minimum')
```

1. Check if x is less than y.
2. If so, print x
3. Otherwise, print y.

Indentation

- Indentation is **important** in Python
  - grouping of statement (block of statements)
  - no explicit brackets, e.g. { }, to group statements

```
x,y = 6,10

if x < y:
    print (x)
else:
    print (y)
    print('the min')
```

skipped

Run the program

6 10

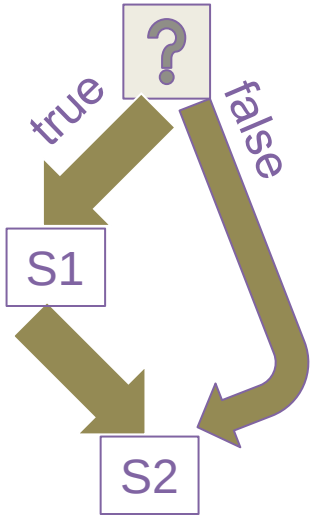
Output

6

if statement (no else!)

- General form of the if statement

```
if boolean-expr :
        S1
S2
```

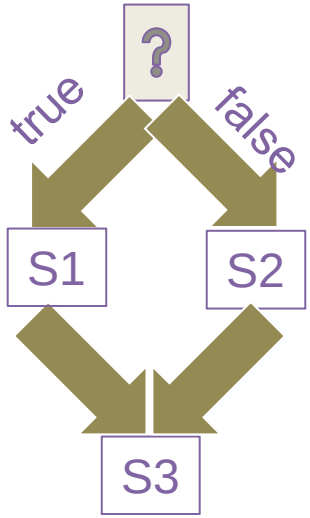


- Execution of if statement
  - First the expression is evaluated.
  - If it evaluates to a **true** value, then S1 is executed and then control moves to the S2.
  - If expression evaluates to **false**, then control moves to the S2 directly.

if-else statement

- General form of the if-else statement

```
if boolean-expr :
        S1
else:
        S2
S3
```



- Execution of if-else statement
  - First the expression is evaluated.
  - If it evaluates to a **true** value, then S1 is executed and then control moves to S3.
  - If expression evaluates to **false**, then S2 is executed and then control moves to S3.
  - S1/S2 can be **blocks** of statements!

Nested if, if-else

```
if a <= b:
        if a <= c:
            …
        else:
            …
else:
        if b <= c) :
            …
        else:
            …
```

Elif

- A special kind of nesting is the chain of if-else-if-else-… statements
- Can be written elegantly using if-elif-..-else

```
if cond1:
        s1
else:
    if cond2:
        s2
    else:
        if cond3:
            s3
        else:
            …
```

```
if cond1:
        s1
elif cond2:
    s2
elif cond3:
    s3
elif  …
else
     last-block-of-stmt
```

Summary of if, if-else

- if-else, nested if's, elif.
- Multiple ways to solve a problem
  - –issues of readability, maintainability
  - –and efficiency

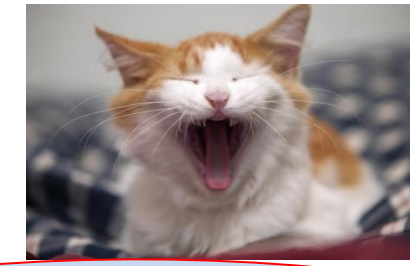
Class Quiz

- What is the value of expression:

(5<2) and (3/0 > 1)

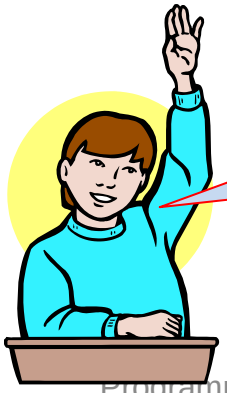
a) Run time crash/error

b) I don't know / I don't care



c) False

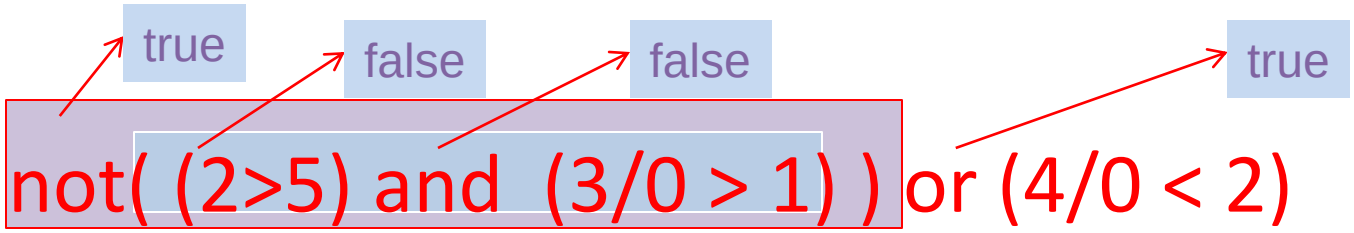
d) True



The correct answer is **False**

Short-circuit Evaluation

- Do not evaluate the second operand of binary short-circuit logical operator if the result can be deduced from the first operand
  - Also applies to nested logical operators



Evaluates to true

3 Factors for Expr Evaluation

- **Precedence**
  - Applied to two different class of operators
  - + and *, - and *, and and or, …
- **Associativity**
  - Applied to operators of same class
  - * and *, + and -, * and /, …
- **Order**
  - Precedence and associativity **identify the operands** for each operator
  - **Not which operand is evaluated first**
  - Python evaluates expressions from left to right
  - While evaluating an assignment, the right-hand side is evaluated before the left-hand side.

Class Quiz

- What is the output of the following program:

```
y = 0.1*3
if y != 0.3:
    print ('Launch a Missile')
else:
    print ("Let's have peace")
```

Launch a Missile

Caution about Using Floats

- Representation of *real numbers* in a computer can not be exact
  - Computers have limited memory to store data
  - *Between any two distinct real numbers, there are infinitely many real numbers.*
- On a typical machine running Python, there are 53 bits of precision available for a Python float

Caution about Using Floats

- The value stored internally for the decimal number 0.1 is the binary fraction

  0.0001100110011001100110011001100110011001100110011001100110011010

- Equivalent to decimal value

  0.1000000000000000055511151231257827021181583404541015625

- Approximation is similar to decimal approximation 1/3 = 0.333333333...
- No matter how many digits you use, you have an approximation

Comparing Floats

- Because of the approximations, comparison of floats is not exact.
- **Solution?**
- Instead of

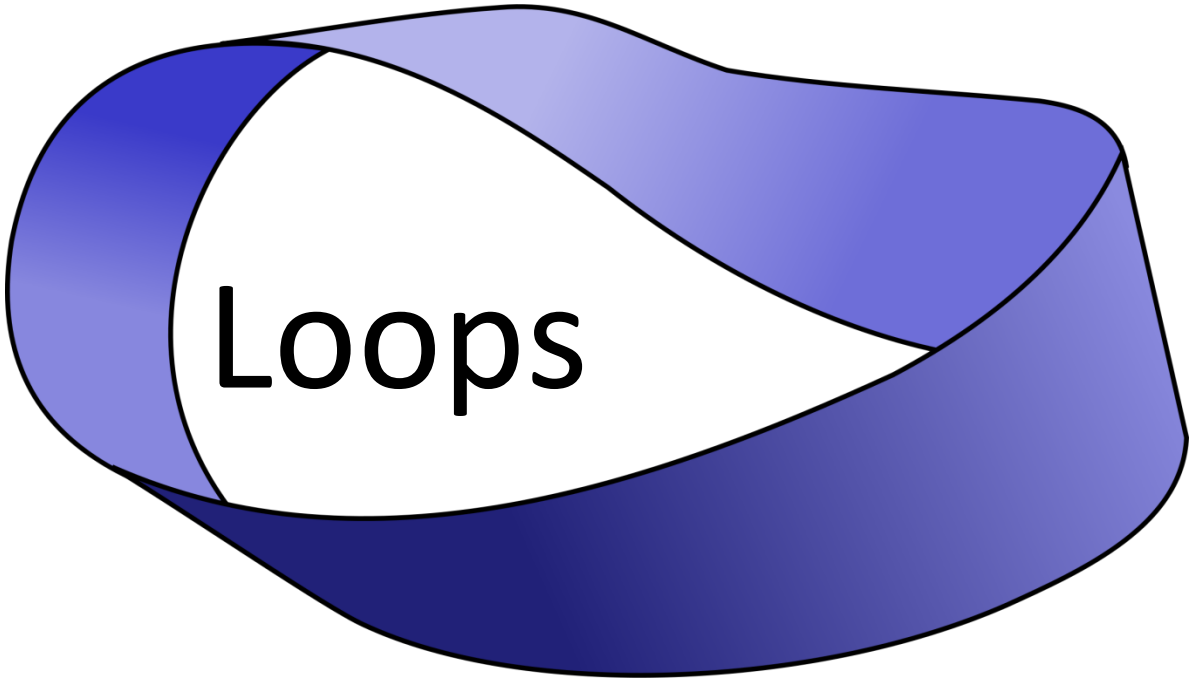
  x == y

  use

  abs(x-y) <= epsilon

  where epsilon is a suitably chosen small value

Programming using Python

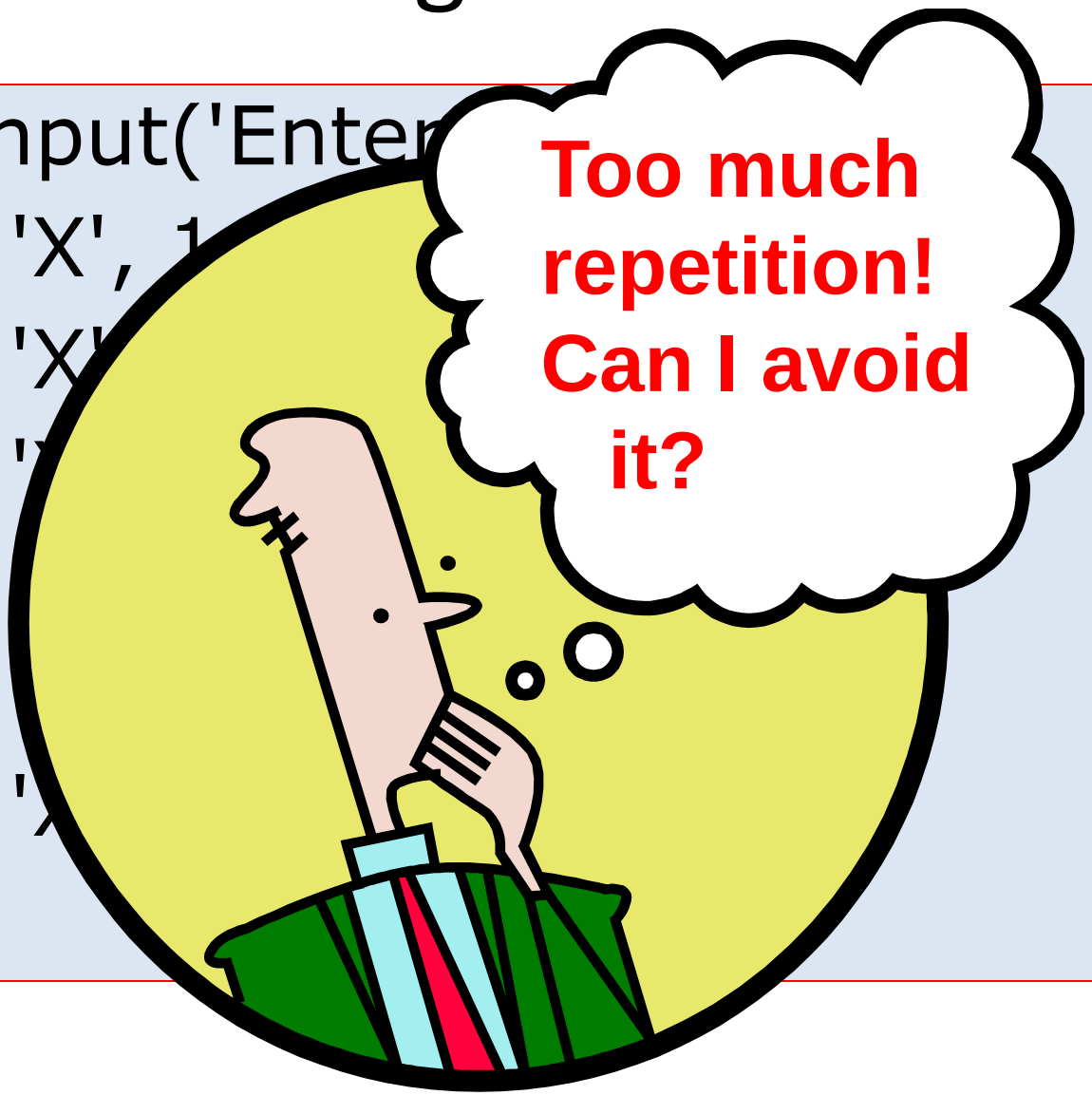


Printing Multiplication Table

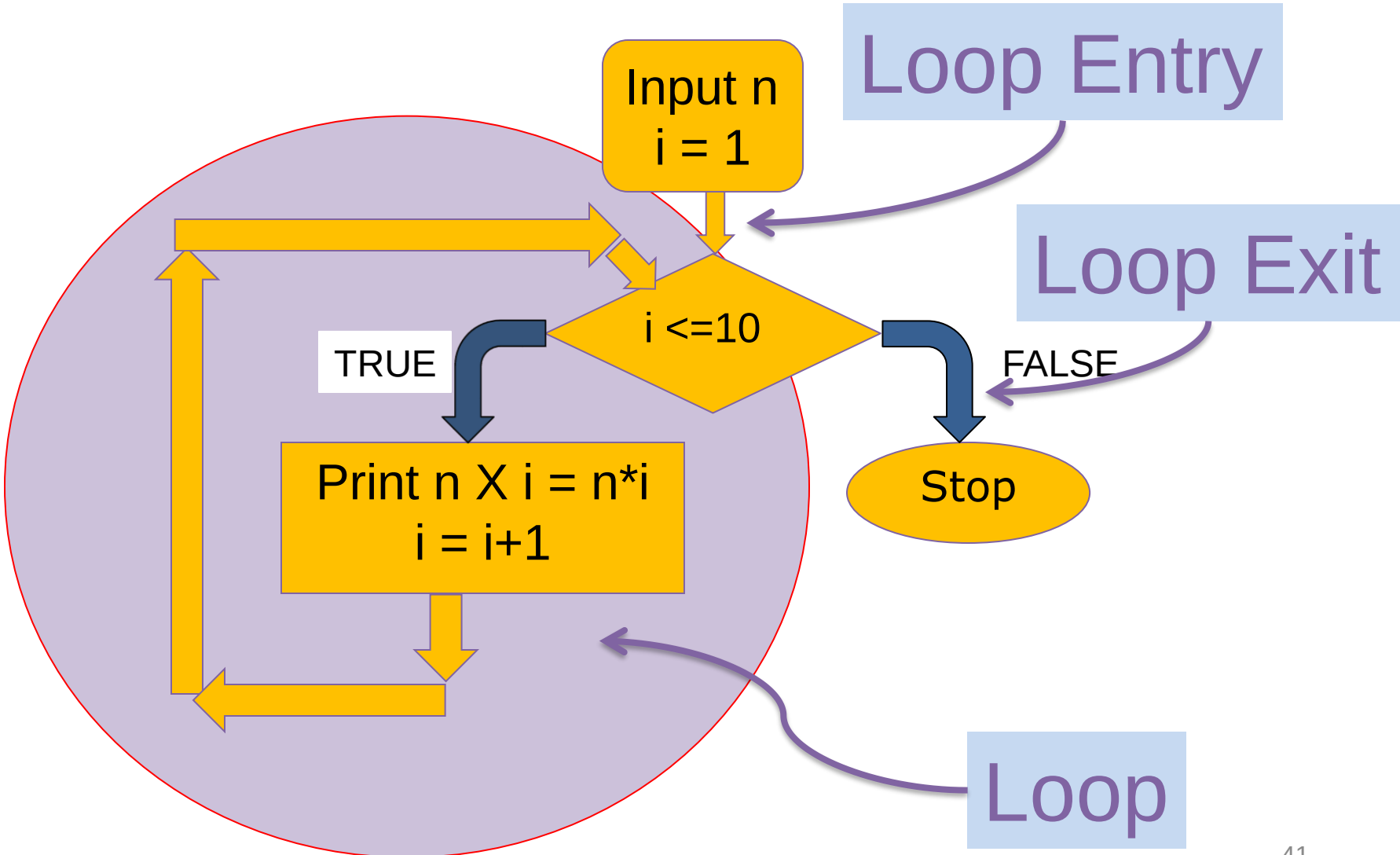
5	X	1	=	5
5	X	2	=	10
5	X	3	=	15
5	X	4	=	20
5	X	5	=	25
5	X	6	=	30
5	X	7	=	35
5	X	8	=	40
5	X	9	=	45
5	X	10	=	50

Program…

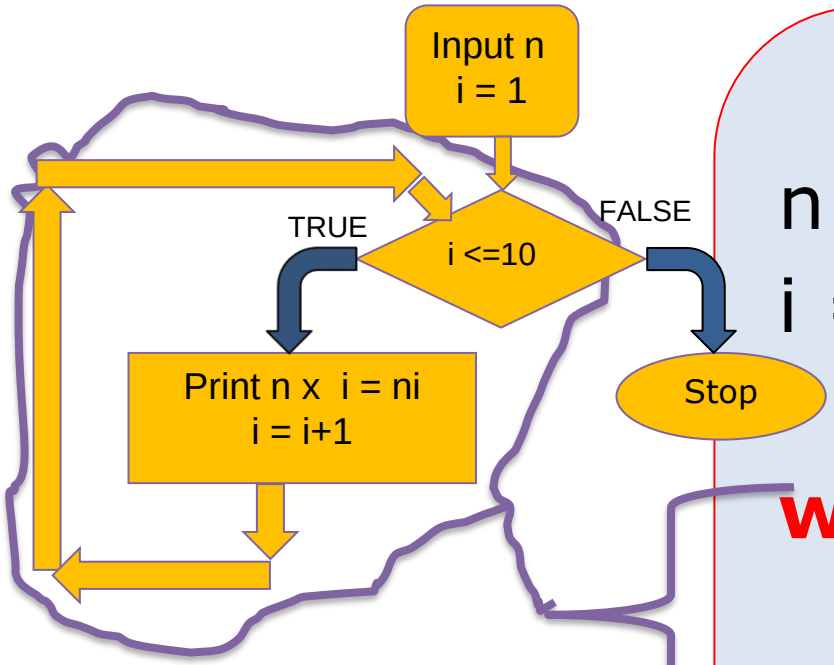
```
n = int(input('Enter
print (n, 'X', 1
print (n, 'X'
print (n, '
print (n,
print (n,
print (n, '
....
```



Printing Multiplication Table



Printing Multiplication Table

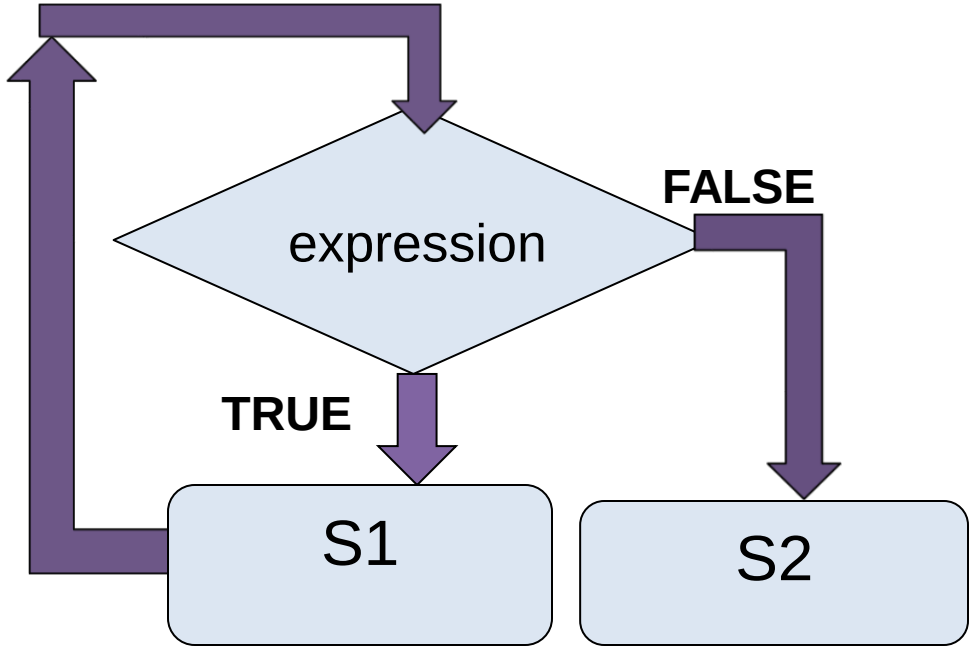



```
n = int(input('n=? '))
i = 1

while (i <= 10) :
    print (n ,'X', i, '=', n*i)
    i = i + 1
print ('done')
```

While Statement

```
while (expression):
        S1
S2
```



1. Evaluate expression
2. If TRUE then
   a) execute statement1
   b) goto step 1.
3. If FALSE then execute statement2.

For Loop

- Print the sum of the reciprocals of the first 100 natural numbers.

```
rsum=0.0# the reciprocal sum

# the for loop
for i in range(1,101):
    rsum = rsum + 1.0/i
print ('sum is', rsum)
```

For loop in Python

- General form

```
for variable in sequence:
    stmt
```

range

- range(s, e, d)
  - generates the list:

    [s, s+d, s+2*d, …, s+k*d]

    where s+k*d < e <= s+(k+1)*d
- range(s, e) is equivalent to range(s, e, 1)
- range(e) is equivalent to range(0, e)

Exercise: What if d is negative? Use python interpreter to find out.

Quiz

- What will be the output of the following program

```
# print all odd numbers < 10
i = 1
while i <= 10:
    if i%2==0: # even
        continue
    print (i, end=' ')
    i = i+1
```

Continue and Update Expr

- Make sure continue does not bypass update-expression for while loops

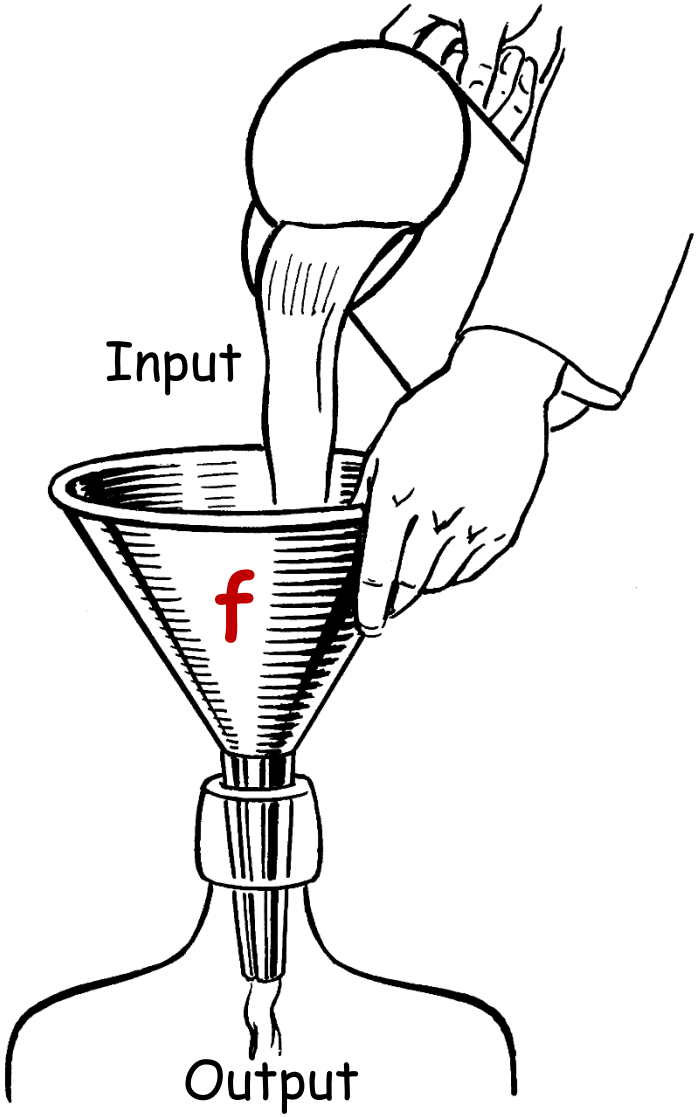
```
# print all odd numbers < 10
i = 1
while i <= 10:
    if i%2==0: # even
        continue
    print (i, end=' ')
    i = i+1
```

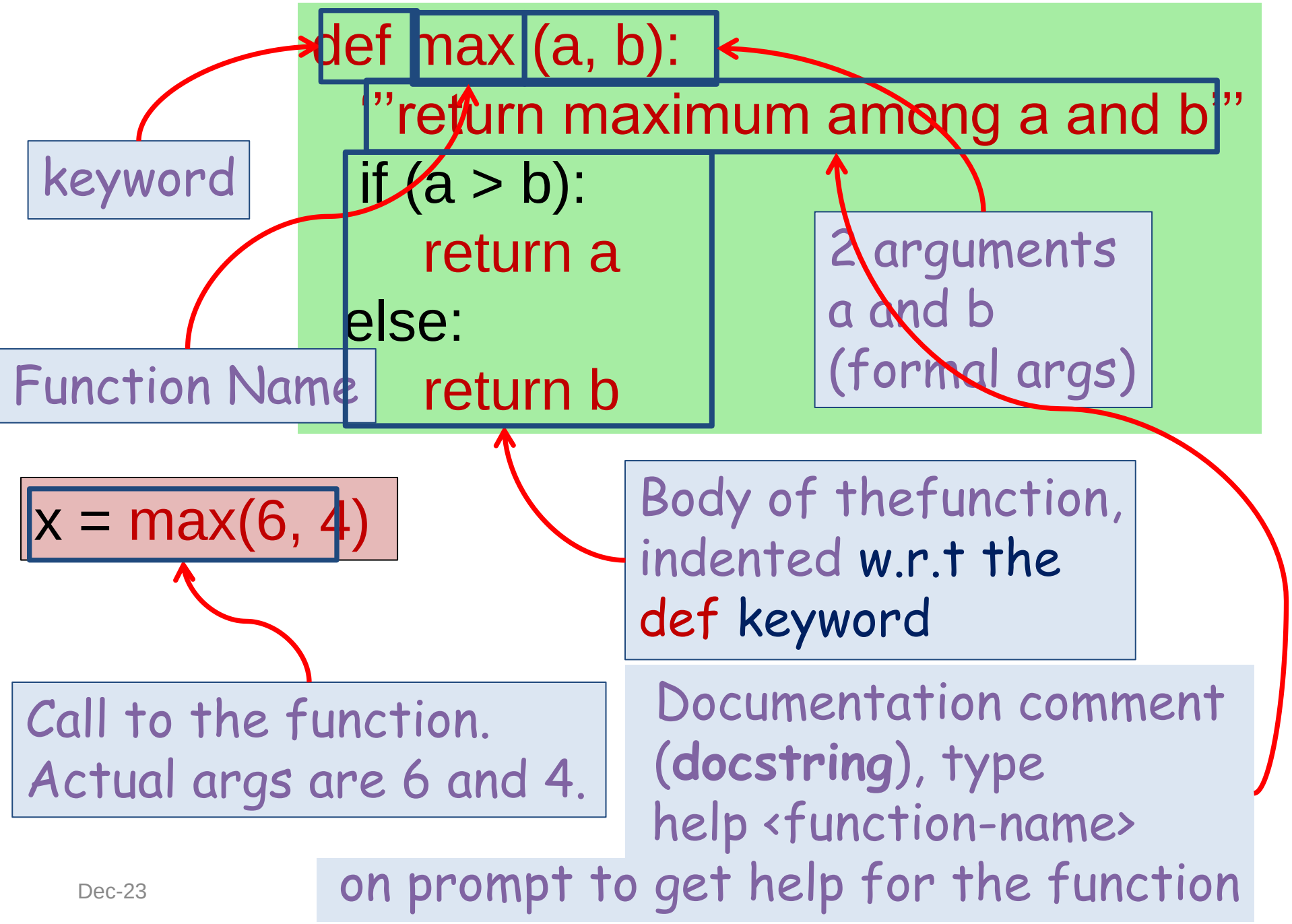
i is not incremented when even number encountered. Infinite loop!!

Programming using Python

f(unctions)

Parts of a function




def max (a, b):
"""return maximum among a and b"""
if (a > b):
return a
else:
return b
keyword
Function Name
2 arguments
a and b
(formal args)
x = max(6, 4)
Body of thefunction,
indented w.r.t the
def keyword
Call to the function.
Actual args are 6 and 4.
Documentation comment
(docstring), type
help <function-name>
on prompt to get help for the function

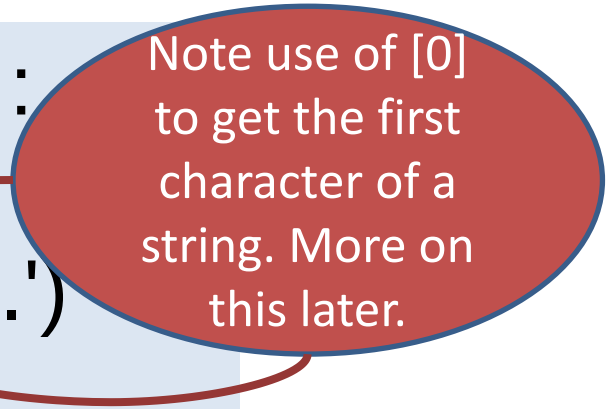
```
def max (a, b):
   “‘return maximum among a and b”’
   if (a > b):
       return a
  else:
       return b
```

```
In[3] : help(max)
Help on function max in module __main__:

max(a, b)
   return maximum among a and b
```

Keyword Arguments

```
def printName(first, last, initials) :
    if initials:
        print (first[0] + '. ' + last[0] + '.')
    else:
        print (first, last)
```



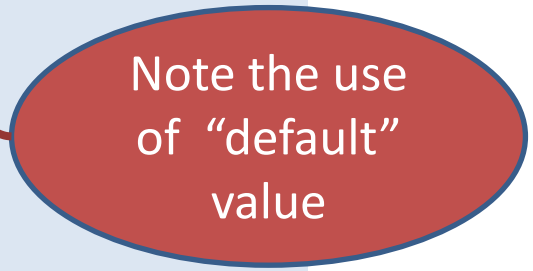
Call	Output
printName('Acads', 'Institute', False)	Acads Institute

Keyword Arguments

- Parameter passing where formal is bound to actual using formal's name
- Can mix keyword and non-keyword arguments
  - All non-keyword arguments precede keyword arguments in the call
  - Non-keyword arguments are matched by position (order is important)
  - Order of keyword arguments is not important

Default Values

```
def printName(first, last, initials=False) :
    if initials:
        print (first[0] + '. ' + last[0] + '.')
    else:
        print (first, last)
```



Call	Output
printName('Acads', 'Institute')	Acads Institute

Default Values

- Allows user to call a function with fewer arguments
- Useful when some argument has a fixed value for most of the calls
- All arguments with default values must be at the end of argument list
  - non-default argument can not follow default argument

Globals

- Globals allow functions to communicate with each other indirectly
  - Without parameter passing/return value
- Convenient when two seemingly “far-apart” functions want to share data
  - No *direct* caller/callee relation
- If a function has to update a global, it must re-declare the global variable with global keyword.

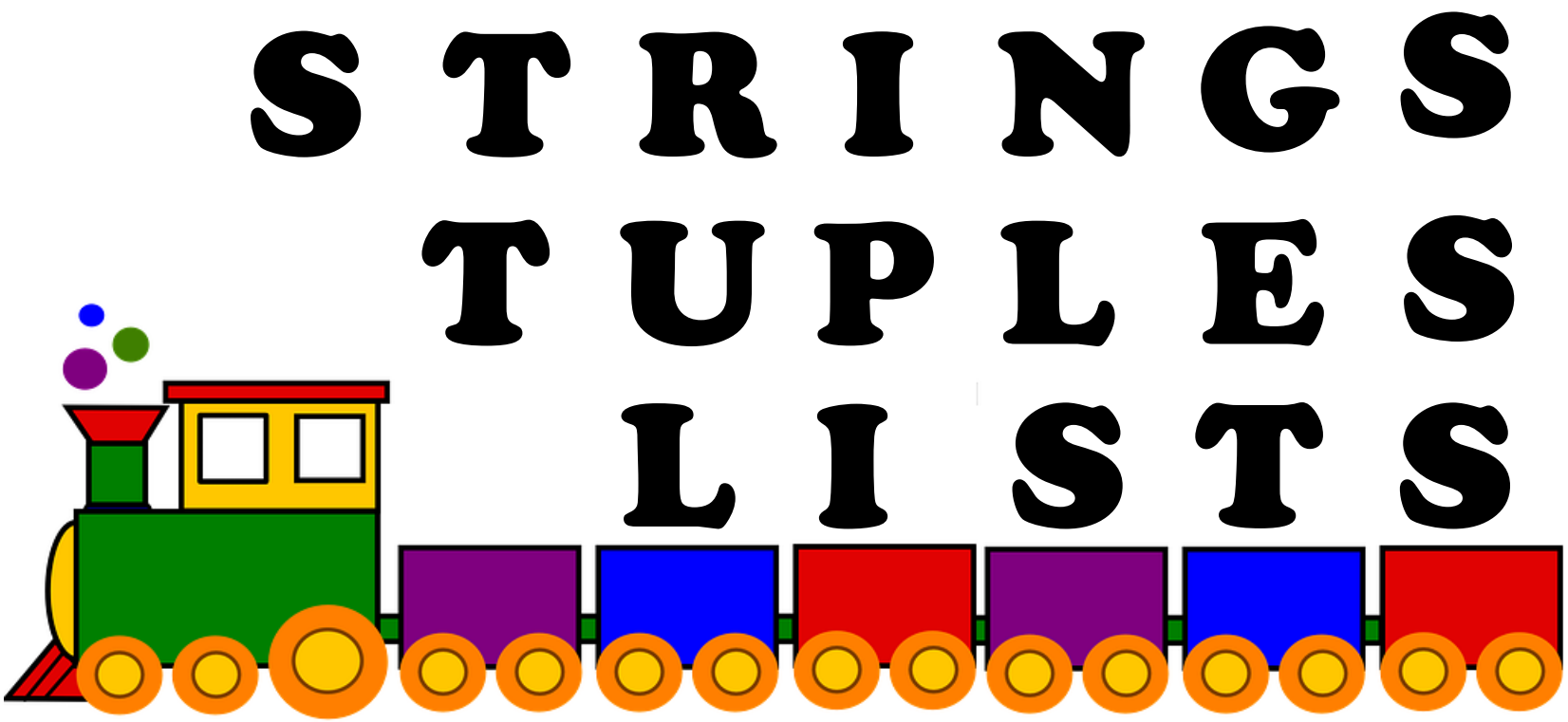
Globals

```
PI = 3.14
def perimeter(r):
    return 2 * PI * r
def area(r):
    return PI * r * r
def update_pi():
    global PI
    PI = 3.14159
```

```
>>> print(area (100))
31400.0
>>> print(perimeter(10))
62.800000000000004
>>> update_pi()
>>> print(area(100))
31415.999999999996
>>> print(perimeter(10))
62.832
```

defines PI to be of float type with value 3.14. PI can be used across functions. Any change to PI in update_pi will be visible to all due to the use of global.

Programming with Python



Strings

- Strings in Python have type str
- They represent sequence of characters
  - Python does not have a type corresponding to character.
- Strings are enclosed in single quotes(') or double quotes(“)
  - Both are equivalent
- Backslash (\) is used to escape quotes and special characters

Strings

```
>>> name='intro to python'
>>> descr='acad\'s first course'
```

- More readable when print is used

```
>>> print descr
acad's first course
```

Length of a String

- len function gives the length of a string

```
>>> name='intro to python'
>>> empty=''
>>> single='a'
```

\n is a **single** character: the special character representing newline

Concatenate and Repeat

- In Python, + and * operations have special meaning when operating on strings
  - + is used for concatenation of (two) strings
  - * is used to repeat a string, an int number of time
  - Function/Operator Overloading

Concatenate and Repeat

```
>>> details = name + ', ' + descr
>>> details
"intro to python, acad's first course"
```

Indexing

- Strings can be indexed
- First character has index 0

```
>>> name='Acads'
```

Indexing

- Negative indices start counting from the right
- Negatives indices start from -1
- -1 means last, -2 second last, ...

```
>>> name='Acads'
>>> name[-1]
's'
>>> name[-5]
'A'
>>> name[-2]
'd'
```

Indexing

- Using an index that is too large or too small results in “index out of range” error

Slicing

- To obtain a substring
- s[start:end] means substring of s starting at index start and ending at index end-1
- s[0:len(s)] is same as s
- Both start and end are optional
  - If start is omitted, it defaults to 0
  - If end is omitted, it defaults to the length of string
- s[:] is same as s[0:len(s)], that is same as s

Slicing

```
>>> name='Acads'
>>> name[0:3]
```

More Slicing

```
>>> name='Acads'
>>> name[-4:-1]
'cad'
>>> name[-4:]
'cads'
>>> name[-4:4]
'cad'
```

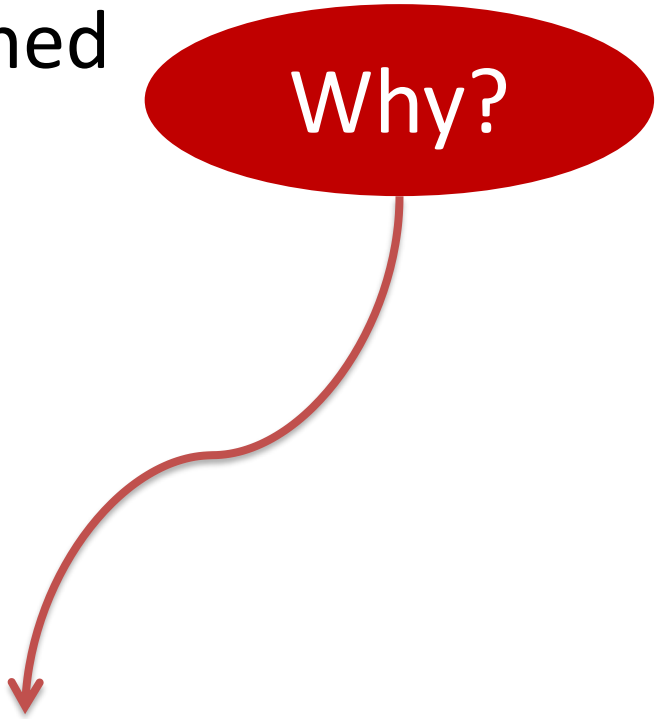
Understanding Indices for slicing

A	c	a	d	s	
0	1	2	3	4	5
-5	-4	-3	-2	-1	

Out of Range Slicing

A	c	a	d	s
0	1	2	3	4
-5	-4	-3	-2	-1

- Out of range indices are ignored for slicing
- when start and end have the same sign, if start >=end, empty slice is returned



Tuples

- A tuple consists of a number of values separated by commas

```
>>> t = 'intro to python', 'amey karkare', 101
```

- Empty and Singleton Tuples

Nested Tuples

- Tuples can be nested

- Note that course tuple is copied into student.
  - Changing course does not affect student

Length of a Tuple

- len function gives the length of a tuple

```
>>> course = 'Python', 'Amey', 101
>>> student = 'Prasanna', 34, course
>>> empty = ()
>>> singleton = 1,
>>> len(empty)
0
>>> len(singleton)
1
>>> len(course)
3
>>> len(student)
3
```

More Operations on Tuples

- Tuples can be concatenated, repeated, indexed and sliced

```
>>> 2*course1
('Python', 'Amey', 101, 'Python', 'Amey', 101)
```

Unpacking Sequences

- Strings and Tuples are examples of sequences
  - Indexing, slicing, concatenation, repetition operations applicable on sequences
- Sequence Unpacking operation can be applied to sequences to get the components
  - *Multiple assignment* statement
  - LHS and RHS must have equal length

Unpacking Sequences

```
>>> student
('Prasanna', 34, ('Python', 'Amey', 101))
>>> name,roll,regdcourse=student
>>> name
```

Lists

- Ordered sequence of values
- Written as a sequence of comma-separated values between square brackets
- Values can be of different types
  - usually the items all have the same type

```
>>> lst = [1,2,3,4,5]
>>> lst
[1, 2, 3, 4, 5]
>>> type(lst)
<type 'list'>
```

Lists

- List is also a sequence type
  - Sequence operations are applicable

Lists

- List is also a sequence type
  - Sequence operations are applicable

```
>>> [0] + fib # Concatenation
```

More Operations on Lists

- L.append(x)
- L.extend(seq)
- L.insert(i, x)
- L.remove(x)
- L.pop(i)
- L.pop()
- L.index(x)
- L.count(x)
- L.**sort**()
- L.**reverse**()

x is any value, seq is a sequence value (list, string, tuple, …),
i is an integer value

Mutable and Immutable Types

- Tuples and List types look very similar
- However, there is one major difference: Lists are **mutable**
  - Contents of a list can be modified
- Tuples and Strings are **immutable**
  - Contents can not be modified

Summary of Sequences

Operation	Meaning
seq[i]	i-th element of the sequence
len(seq)	Length of the sequence
seq1 + seq2	Concatenate the two sequences
num*seq seq*num	Repeat seq num times
seq[start:end]	slice starting from **start**, and ending at **end-1**
e in seq	True if e is present is seq, False otherwise
e not in seq	True if e is not present is seq, False otherwise
for e in seq	Iterate over all elements in seq (e is bound to one element per iteration)

Sequence types include String, Tuple and List.
Lists are mutable, Tuple and Strings immutable.

Summary of Sequences

- For details and many useful functions, refer to:

https://docs.python.org/3.2/tutorial/datastructures.html

Programming with Python

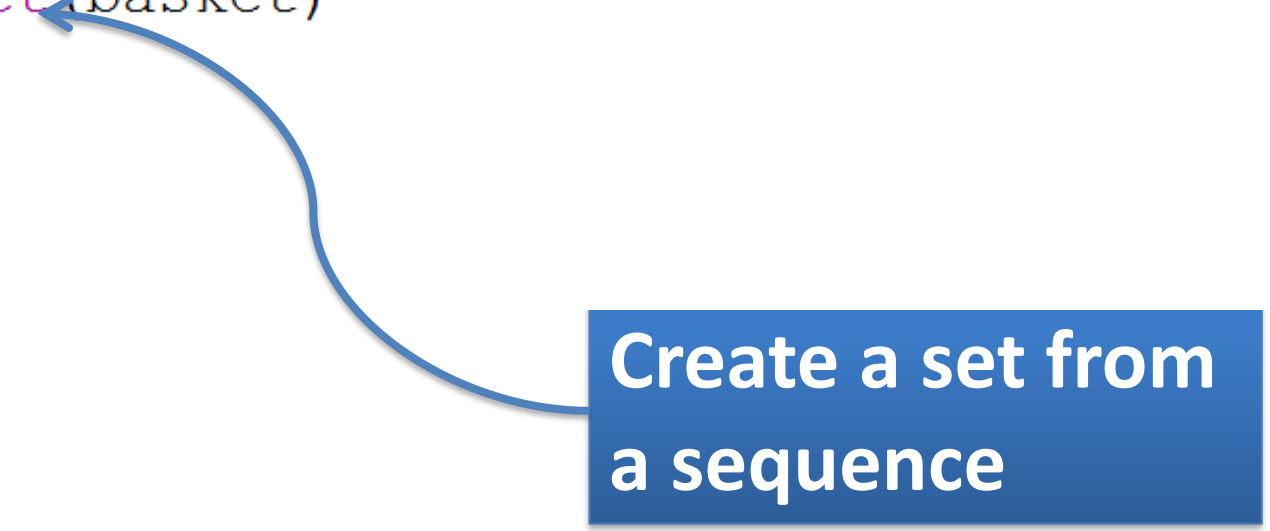
Sets and Dictionaries

Sets

- An unordered collection with no duplicate elements
- Supports
  - membership testing
  - eliminating duplicate entries
  - Set operations: union, intersection, difference, and symmetric difference.

Sets

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'o
range', 'banana']
>>> fruits = set(basket)
```



Set Operations

```
>>> A=set('acads')
>>> B=set('institute')
>>> A
    {'a', 's', 'c', 'd'}
>>> B
    {'e', 'i', 'n', 's', 'u', 't'}
```

Dictionaries

- Unordered set of *key:value* pairs,
- Keys have to be unique and immutable
- Key:value pairs enclosed inside curly braces {...}
- Empty dictionary is created by writing {}
- Dictionaries are mutable
  - add new key:value pairs,
  - change the pairing
  - delete a key (and associated value)

Operations on Dictionaries

Operation	Meaning
len(d)	Number of key:value pairs in d
d.keys()	List containing the keys in d
d.values()	List containing the values in d
k in d	True if key k is in d
d[k]	Value associated with key k in d
d.get(k, v)	If k is present in d, then d[k] else v
d[k] = v	Map the value v to key k in d (replace d[k] if present)
del d[k]	Remove key k (and associated value) from d
for k in d	Iterate over the keys in d

Operations on Dictionaries

```
>>> capital = {'India':'New Delhi', 'USA':'Washingto
n DC', 'France':'Paris', 'Sri Lanka':'Colombo'}
```

Operations on Dictionaries

Operations on Dictionaries

Dictionary Construction

- The dict constructor: builds dictionaries directly from *sequences of key-value pairs*

```
>>> airports=dict([('Mumbai', 'BOM'), ('Delhi', 'Del
'),('Chennai', 'MAA'), ('Kolkata', 'CCU')])
>>> airports
{'Kolkata': 'CCU', 'Chennai': 'MAA', 'Delhi': 'Del',
'Mumbai': 'BOM'}
```

Programming with Python

File I/O

File I/O

- Files are persistent storage
- Allow data to be stored beyond program lifetime
- The basic operations on files are
  - open, close, read, write
- Python treat files as sequence of lines
  - sequence operations work for the data read from files

File I/O: open and close

open(filename, mode)

- While opening a file, you need to supply
  - The name of the file, including the path
  - The mode in which you want to open a file
  - Common modes are r (read), w (write), a (append)
- Mode is optional, defaults to r
- open(..) returns a file object
- close() on the file object closes the file
  - finishes any buffered operations

File I/O: Example

```
>>> players = open('tennis_players', 'w')
>>>
>>>
>>>
>>>
>>>
>>> players.close() # done with writing
```

- Do some writing
- How to do it?
  - see the next few slides

File I/O: read, write and append

- Reading from an open file returns the contents of the file
  - as **sequence** of lines in the program
- Writing to a file
  - IMPORTANT: If opened with mode 'w', **clears** the existing contents of the file
  - Use append mode ('a') to preserve the contents
  - Writing happens at the end

File I/O: Examples

```
>>> players = open('tennis_players', 'w')
```

```
>>> players.close() # done with writing
```

File I/O: Examples

```
>>> print(players)
```

```
>>> pn = n.read() # read all players
```

File I/O: Examples

```
>>> n = open('tennis_players', 'r')
>>> c = open('tennis_countries', 'r')
```

of for ... in

File I/O: Examples

Programming using Python

Modules and Packages

Amey Karkare

Dept. of CSE

IIT Kanpur

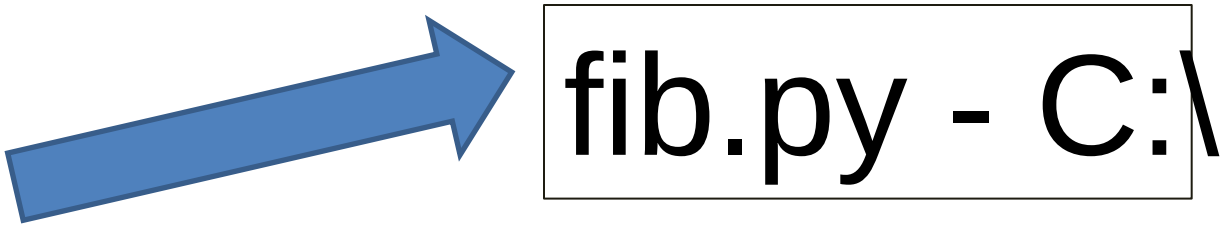
Modules

- As program gets longer, need to organize them for easier access and easier maintenance.
- Reuse same functions across programs without copying its definition into each program.
- Python allows putting definitions in a file
  - use them in a script or in an interactive instance of the interpreter
- Such a file is called a *module*
  - definitions from a module can be *imported* into other modules or into the *main* module

Modules

- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix .py appended.
- Within a module, the module's name is available in the global variable __name__.

Modules Example



fib.py - C:\Users\karkare\Google Drive\IITK\Courses\2016Python\Programs\fib.py (2.7.12)

File Edit Format Run Options Window Help

```
# Module for fibonacci numbers

def fib_rec(n):
    '''recursive fibonacci'''
    if (n <= 1):
        return n
    else:
        return fib_rec(n-1) + fib_rec(n-2)
```

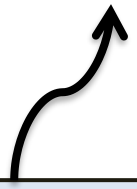
Modules Example

```
def fib_rec(n):
    '''recursive fibonacci'''
    if (n <= 1):
        return n
    else:
        return fib_rec(n-1) + fib_rec(n-2)

def fib_iter(n):
    '''iterative fibonacci'''
    cur, nxt = 0, 1
    for k in range(n):
        cur, nxt = nxt, cur+nxt
    return cur

def fib_upto(n):
    '''given n, return list of fibonacci
    numbers <= n'''
    cur, nxt = 0, 1
    lst = []
    while (cur < n):
        lst.append(cur)
        cur, nxt = nxt, cur+nxt
    return lst
```

```
>>> import fib
>>> fib.fib_upto(5)
[0, 1, 1, 2, 3]
>>> fib.fib_rec(10)
55
>>> fib.fib_iter(20)
6765
>>> fib.__name__
'fib'
```



Within a module, the module's name is available as the value of the global variable __name__.

Importing Specific Functions

- To import specific functions from a module

- This brings only the imported functions in the current symbol table
  - No need of modulename. (absence of `fib.` in the example)

Importing ALL Functions

- To import *all* functions from a module, in the current symbol table

```
>>> from fib import *
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(8)
21
```

- This imports all names except those beginning with an underscore (_).

`__main__` in Modules

- When you run a module on the command line with

  `python fib.py <arguments>`

  the code in the module will be executed, just as if you imported it, but with the `__name__` set to "__main__".
- By adding this code at the end of your module

```
if __name__ == "__main__":
    ... # Some code here
```

  you can make the file usable as a script as well as an importable module

`__main__` in Modules

```
if __name__ == "__main__":
    import sys
    print (fib_iter(int(sys.argv[1])))
```

- This code parses the command line only if the module is executed as the “main” file:

```
$ python fib.py 10
55
```

- If the module is imported, the code is not run:

```
>>> import fib
>>>
```

Package

- A Python package is a collection of Python modules.
- Another level of *organization.*
- *Packages* are a way of structuring Python's module namespace by using *dotted module names*.
  - The module name A.B designates a submodule named B in a package named A.
  - The use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other's module names.

A **sound** Package

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

https://docs.python.org/3/tutorial/modules.html

A **sound** Package

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

What are these files with funny names?

https://docs.python.org/3/tutorial/modules.html

`__init.py__`

- The `__init__.py` files are required to make Python treat directories containing the file as packages.
- This prevents directories with a common name, such as `string`, unintentionally hiding valid modules that occur later on the module search path.
- `__init__.py` can just be an empty file
- It can also execute initialization code for the package

Importing Modules from Packages

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

https://docs.python.org/3/tutorial/modules.ht

Importing Modules from Packages

`import sound.effects.echo`

- Loads the submodule `sound.effects.echo`
- It must be referenced with its full name:

```
sound.effects.echo.echofilter(
    input, output,
    delay=0.7, atten=4
)
```

Importing Modules from Packages

```
from sound.effects import echo
```

- This also loads the submodule `echo`
- Makes it available without package prefix
- It can be used as:

```
echo.echofilter(
    input, output,
    delay=0.7, atten=4
)
```

Importing Modules from Packages

```
from sound.effects.echo import echofilter
```

- This loads the submodule `echo`, but this makes its function `echofilter()` directly available.

```
echofilter(input, output,
           delay=0.7, atten=4)
```

Popular Packages

- pandas, numpy, scipy, matplotlib, …
- Provide a lot of useful functions

HTML CONTENTS

- INTRODUCTION OF HTML
- OBJECTIVE OF HTML
- WORLD WIDE WEB
- HTML TOOLS
- HTML TERMINOLGY
- HOW TO CREATE AN HTML DOCUMENT
- SAVING AND VIEWING A HTML DOCUMENT
- TEXT TEGS
- SPECIAL CHARTACTER
- ADVANTAGES OF HTML
- DISADVANTAGES OF HTML

INTRODUCTION OF HTML

- HTML is a language for describing web pages.
- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is not a programming language, it is a **markup language**
- A markup language is a set of **markup tags**
- HTML uses **markup tags** to describe web pages

INTRODUCTION OF HTML

- HTML (Hypertext Markup Language) is used to create document on the World Wide Web. It is simply a collection of certain key words called 'Tags' that are helpful in writing the document to be displayed using a browser on Internet.

  It is a platform independent language that can be used on any platform such as Windows, Linux, Macintosh, and so on. To display a document in web it is essential to mark-up the different elements (headings, paragraphs, tables, and so on) of the document with the HTML tags. To view a mark-up document user has to open the document in a browser. A browser understands and interpret the HTML tags, identifies the structure of the document (which part are which) and makes decision about presentation (how the parts look) of the document.

  HTML also provides tags to make the document look attractive using graphics, font size and colors. User can make a link to the other document or the different section of the same document by creating Hypertext Links also known as Hyperlinks

OBJECTIVE OF HTML

- create, save and view a HTML document
- format a web page using section heading tags
- describe Ordered and Unordered lists
- explain graphics in HTML document
- describe hypertext links and making text/image link

WORLD WIDE WEB

- The **World Wide Web** (abbreviated as **WWW** or **W3** and commonly known as **the Web)**is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and navigate between them via hyperlinks.

HTML TOOLS

- There are two tools of HTML.

a) HTML Editor: it is the program that one uses to create and save HTML documents. They fall into two categories:

- Text based or code based which allows one to see the HTML code as one is creating a document.e.g. Notepad.

- Netscape composer

HTML TOOLS

b) Web Browser: it is the program that one uses to view and test the HTML documents. They translate Html encoded files into text,image,sounds and other features user see. Microsoft Internet Explorer,Netscape,Mosaic Chrome are examples of browsers that enables user to view text and images and many more other World Wide Web featueres.They are software that must be installed on user computer.

HTML TERMINOLGY

- Some commonly used terms in HTML are:
- a)Tag: Tags are always written within angles brackets. it is a piece of text is used to identify an element so that the browser realizes how to display its contents.e.g.<HTML> tag indicates the start of an HTML document .HTML tag can be two types. They are:-
- -Paired Tags :A tag is said to be a paired tag if text is placed between a tag and its companions tag.In paired tag ,the first tag is referred to as opening tag and the second tag is referred to as closing tag.
- -Unpaired Tags: An unpaired tag does not have a companion tag .unpaired tag also known as singular or Stand-Alone tags.e.g:
,<hr> etc.

HTML TERMINOLGY

b) Attribute: Attribute is the property of an tag that specified in the opening angle brackets. It supplies additional information like color,size,home font-style etc to the browser about a tag. E.g. most of the common attributes are height, color,width,src,border,align etc.

c) DTD: Document Type Definition is a collection of rules written in standard Generalized Markup Language(SGML).HTML is define in terms of its DTDS. All the details of HTML tags, entities and related document structure are defined in the DTDS.

d) ELEMENT: Element is the component of a document's structure such as a title, a paragraph or a list. It can include an opening and a closing tag and the contents within it.

HOW TO CREATE AN HTML DOCUMENT

- The essential tags that are required to create a HTML document are:
- <HTML>.....</HTML>
- <HEAD>.....</HEAD>
- <BODY>.....</BODY>

HTML Tag <HTML>

- The <HTML> tag encloses all other HTML tags and associated text within your document. It is an optional tag. You can create an HTML document that omits these tags, and your browser can still read it and display it. But it is always a good form to include the start and stop tags.The format is:
- <HTML>

Your Title and Document (contains text with HTML tags) goes here

- </HTML>

Most HTML tags have two parts, an opening tag and closing tag. The closing tag is the same as the opening tag, except for the slash mark e.g. </HTML>. The slash mark is always used in closing tags.

An HTML document has two distinct parts HEAD and BODY

- <HTML>
- <HEAD>
- .....
- .....
- .....
- </HEAD>
- <BODY>
- .....
- .....
- .....
- </BODY>
- </HTML>

HEAD Tag <HEAD>

- HEAD tag comes after the HTML start tag. It contains TITLE tag to give the document a title that displays on the browsers title bar at the top. The Format is:

<HEAD>

<TITLE>

Your title goes here

</TITLE>

</HEAD>

BODY Tag <BODY>

- The BODY tag contains all the text and graphics of the document with all the HTML tags that are used for control and formatting of the page.The Format is:

<BODY>
Your Document goes here
</BODY>

An HTML document, web page can be created using a text editor, Notepad or WordPad. All the HTML documents should have the extension .htm or html. It require a web browser like Internet Explorer or Netscape Navigator/Communicator to view the document.

Attributes used with <BODY>

- BGCOLOR: used to set the background color for the document Example:

<BODY BGCOLOR="yellow">

Your document text goes here.

</BODY>

- TEXT: used to set the color of the text of the document Example:

<BODY TEXT="red">Introduction to HTML:: 77

Document text changed to red color

</BODY>

Document text changed to red color

Attributes used with <BODY>

- MARGINS: set the left hand/right hand margin of the document LEFTMARGIN: set the left hand margin of the document Example:

<BODY LEFTMARGIN="60">

This document is indented 60 pixels from the left hand side

of the page.

</BODY>

- TOPMARGIN: set the left hand margin of the document Example:

<BODY TOPMARGIN="60">

This document is indented 60 pixels from the top of the page.

</BODY>

Attributes used with <BODY>

- BACKGROUND: It is used to point to an image file (the files with an extension .gif, .jpeg) that will be used as the background of the document. The image file will be tiled across the document. Example:

<BODY BACKGROUND="filename. if">

Your document text goes here

</BODY>

Follow the steps to create and view in browser

- Step-1: Open text editor Notepad (click on Start→ All Programs→ Accessories Notepad)
- Step-2: Enter the following lines of code:

```
<HTML>
<HEAD>
<TITLE>
My first Page
</TITLE>
</HEAD>
<BODY>
WELCOME TO MY FIRST WEB PAGE
</BODY>
</HTML>
```

SAVING AND VIEWING A HTML DOCUMENT

- Step-3: Save the file as myfirstpage.html (go to File-Save As give File name: myfirstpage.html-choose save as type: All Files-click save)
- Step-4: Viewing document in web browser (open Internet Explorer-click on File-Open-Browse-select the file myfirstpage.html-click open-click ok

TEXT TEGS

- Text tag are dividing into two categories as:

-Character-level tags and attributes which applies to formatting of individual letters or words.

-Paragraph level tags and attributes which apply

=To formatting of sections of text.

Character Formatting Tag

- The character formatting tags are used to specify how a particular text should be displayed on the screen to distinguish certain characters within the document.

The most common character formatting tags are

- Boldface <B>: displays text in BOLD

Example: Welcome to the <B> Internet World </B>

Output: Welcome to the Internet World

- Italics <I>: displays text in Italic

Example: Welcome to the <I> Internet World </I>

Output: Welcome to the Internet World

- Subscript <SUB>: displays text in Subscript
- Superscript <SUP>: displays text in Superscript
- Small <SMALL>: displays text in smaller font as compared to normal font
- Big <BIG>: displays text in larger font as compared to normal font
- Underline<U>specifies that the enclosed text be underline

Example:<U> hello</u>

Output: <u>hello</u>

Font Colors and Size:<FONT>

- By using <FONT> Tag one can specify the colors, size of the text. Example:

<FONT> Your text goes here </FONT>

Attributes of <FONT> are:

- COLOR: Sets the color of the text that will appear on the

screen. It can be set by giving the value as #rr0000 for red (in RGB hexadecimal format), or by name. Example: <FONT COLOR="RED"> Your text goes here </FONT>

Font Colors and Size:<FONT

- SIZE: Sets the size of the text, takes value between 1 and 7, default is 3. Size can also be set relative to default size for example; SIZE=+X, where X is any integer value and it will add with the default size.
- Example:

<FONT SIZE=5> Font Size changes to 5 </FONT>

- FACE: Sets the normal font type, provided it is installed on the user's machine.
- Example:
- <FONT FACE="ARIAL"> the text will be displayed in Arial</FONT>

An HTML document formatText.html shows the use of Character Formatting Tags.

```
<HTML>
<HEAD>
<TITLE>
Use of Character Formatting Text Tags
</TITLE>
</HEAD>
<BODY>
<H1><I> Welcome to the world of Internet</I></H1>
 It is a
<FONT COLOR="BLUE" SIZE="4">
<U>Network of Networks</U>
</FONT>
</BODY>
</HTML>
```

OUTPUT

Welcome to the world of Internet

It is a Network of Networks

MARQUEE TAG

- This tag is used text horizontally across the screen.it is mainly used to deliver a specfic message to the visitor or to scroll Ads on a page.
- Example: <marquee> hello world></marquee>

Attributes of marquee tag

- Bgcolor : Sets the background color of the marquee.
- Direction :Sets the direction of the marquee box to either left-to-right, right-to-left, up-to-down and down-to-up.
- Width: This sets how wide the marquee should be.
- Loop: This sets how many times the marquee should 'Loop' its text. Each trip counts as one loop.

paragraph Formatting Tag

- Paragraph level formatting applies to formatting of an entire portion of text unlike character level tags where only individual letters or words are formatted.

The most common paragraph formatting tags are

- Using paragraph tag: <P>

This tag<P>indicates a paragraph,used to separate two paragraphs with a blank line.

- Example:

<P> Welcome to the world of HTML </P>

<P> First paragraph. Text of First paragraph goes here</P>

- Output:

Welcome to the world of HTML

First paragraph. Text of First paragraph goes her

Using Line Break Tag:

- The empty tag
 is used, where the text needs to start from a new line and not continue on the same line. To get every sentence on a new line, it is necessary to use a line break.
- Example:

<BODY>National Institute of Open Schooling

B-31B, Calipash Colony

New Delhi-110048</BODY>

- Output:

National Institute of Open Schooling

B-31B, Calipash Colony

New Delhi-11004

Using Preformatted Text Tag: <PRE>

- <PRE> tag can be used, where it requires total control over s p a c i n g and l i n e b r e a k s such as t y p i n g a p o e m. B r o w s e r preserves your space and line break in the text written inside the tag.
- Example:

```
<PRE>
National Institute of Open Schooling
        B-31B, Kailash Colony
New Delhi-110048
</PRE>
```

- Output:

National Institute of Open Schooling
        B-31B, Kailash Colony
New Delhi-11004

An HTML document control.html shows the use of <P>,
 and <PRE>

<HTML>
<HEAD>
<TITLE>
Use of Paragraph, Line break and preformatted text Tag
</TITLE>
</HEAD>
<BODY>
HTML Tutorial
<P>
HTML stands for Hypertext Markup Language
It is used for creating web page. It is very simple
and easy to learn.

An HTML document control.html shows the use of <P>,
 and <PRE>

```
</P>
<P>
HTML stands for Hypertext Markup Language.<BR>
It is used for creating web page. It is very simple<BR>
and easy to learn.<BR>
</P>
<PRE>
HTML stands for Hypertext Markup Language
It is used for creating web page. It is very simple
and easy to learn.
</PRE>
</BODY>
 </HTML>
```

OUTPUT

- HTML Tutorial

HTML stands for Hypertext Markup Language. It is used for creating web page. It is very simple and easy to learn.

HTML stands for Hypertext Markup Language.
It is used for creating web page. It is very simple
and easy to learn.

HTML stands for Hypertext Markup Language.
It is used for creating web page. It is very simple
and easy to learn.

Using Horizontal Rule Tag: <HR>

- An empty tag <HR> basically used to draw lines and horizontal rules. It can be used to separate two sections of text.
- Example:

<BODY>

Your horizontal rule goes here. <HR>

The rest of the text goes here.

</BODY>

- Output:

Your horizontal rule goes here.

The rest of the text goes her

<HR> accepts following attributes

- SIZE: Determines the thickness of the horizontal rule. The value is given as a pixel value.

Example: <HR SIZE="3">

- WIDTH: Specifies an exact width of HR in pixels, or arelative width as percentage of the document width.

Example: <HR WIDTH="50%">, horizontal rule a width a 50 percent of the page width.

- ALIGN: Set the alignment of the rule to LEFT, RIGHT and CENTER. It is applicable if it is not equal to width of the page.
- NOSHADE: If a solid bar is required, this attribute is used; it specifies that the horizontal rule should not be shaded at all.
- COLOR: Set the color of the Horizontal rule.

Example: <HR COLOR="BLUE“>

Example of <HR> with its attribute:

<HR ALIGN=' 'CENTER' ' WIDTH=' '50%' ' SIZE=' '3" NOSHADE
COLOR="BLUE“>

HEADING: <H1>.....<H6>tags

HTML has six header tags <H1>, <H2>.....<H6> used to specify section headings. Text with header tags is displayed in larger and bolder fonts than the normal body text by a web browser. Every .header leaves a blank line above and below it when displayed in browse.

Example: An HTML document, headings.html shows the different section headings

```
.
<HTML>
<HEAD>
<TITLE>
Section Heading
</TITLE>
</HEAD>
<BODY>
<H1> This is Section Heading 1 </H1>
<H2> This is Section Heading 2 </H2>
<H3> This is Section Heading 3 </H3>
<H4> This is Section Heading 4 </H4>
<H5> This is Section Heading 5 </H5>
<H6> This is Section Heading 6 </H6>
</BODY>
</HTML>
```

Viewing output of HTML document headings.html in browse

This is Section Heading 1

This is Section Heading 2

This is Section Heading 3

This is Section Heading 4

This is Section Heading 5

This is Section Heading 6

SPECIAL CHARTACTER

- There are certain special characters that can be used while creating document.Following are some special character:
- Symbols Entity

©, ®	&copy, &reg
¼, ½, ¾	&frac14, &frac12, &frac34
÷, <, >, ≤,≥	&divide, &lt, &gt, &le, &ge
&	&amp
♣ ♠ ♥	&spades, &clubs, &hearts

All these special character must be ended with a semicolon;

Example:

<PRE>

The copyright symbol is: ©

The registered rank is: ®

</PRE>

- Output:

The copyright symbol is:©

The registered rank is:®

ADVANTAGES OF HTML

- Easy to use
- Loose syntax (although, being too flexible will not comply with standards).
- Supported on almost every browser, if not all browsers.
- Widely used; established on almost every website, if not all websites.
- Very similar to XML syntax, which is increasingly used for data storage.
- Free - You need not buy any software.
- Easy to learn & code even for novice programmers.

DISADVANTAGES OF HTML

- It cannot produce dynamic output alone, since it is a static language
- Sometimes, the structuring of HTML documents is hard to grasp
- You have to keep up with deprecated tags, and make sure not to use them
- Deprecated tags appear because another language that works with HTML has replaced the original work of the tag; thus the other language needs to be learned (most of the time, it is CSS)
- Security features offered by HTML are limited

PRESENTATION SUMMARY

-> What is CSS ?

-> CSS and HTML

-> The Box Model.

-> Style Sheet Implementation.

-> CSS Rule Structure.

-> HTML and DIV's.

-> Common CSS properties.

WHAT IS CSS ?

Css stands for cascading style sheet. It is not a language. It is a pat of design. CSS is a heart of HTML. Typical CSS file is a text file with an extension ".CSS" and contain a series of commands.

HTML WITHOUT CSS:-

"Html without CSS is like a piece of candy without a pretty wrapper."

Without CSS, HTML elements typically flow from top to bottom of the page and position themselves to the left by default .

With CSS help, we can create containers or DIV's to better organize content and make a Web page visually appealing.

CSS AND HTML:-

(a)HTML and CSS work together to produce beautiful and functional Web sites.

(b) HTML= Structure

(c) CSS= Style

THE BOX MODEL:-

CSS works on the box model. A typical Web page consists of many boxes joined together from top to bottom . These boxes can be stacked nested, and float.

ATTACHING A STYLE SHEET:-

Attach a style sheet to a page by adding the code to the <head>

Section of the HTML page. There are 3 ways to attach CSS to a page:

1.External style sheet:- *Best used to control styling on multiple pages.*

```
<link href="css/style.css" type="text/css"
  rel="stylesheet"/>
```

2.Internal style sheet:- *Best used to control styling in the page.*

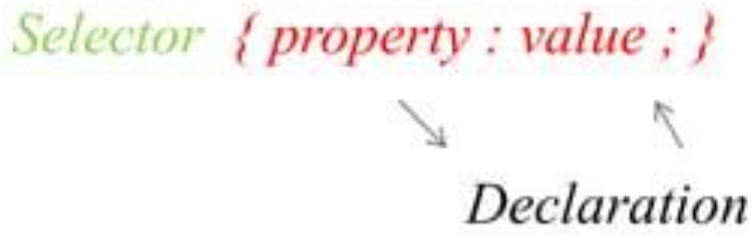
```
<style type="type="text/css">
H1 { color : red }
</style>
```

3.Inline Style Sheet:- *CSS is not attached in the <header> but is used directly within HTML tages.*

```
<p style="color : red"> Some Text </p>
```

CSS RULE STRUCTURE:-

A CSS RULE is made up of a selector and a declaration. A declaration consists of property and value.



SELECTOR:-

A selector, here in green, is often an element of HTML.

```
body { property : value; }
h1 { property : value; }
em { property : value; }
p { property : value; }
```

PROPERTIES AND VALUES:-

```
body { background ; purple; }
  h1   { color :  green;  }
  h2   { font-size:  large ;  }
   p   { color :  #FFF ; }
```

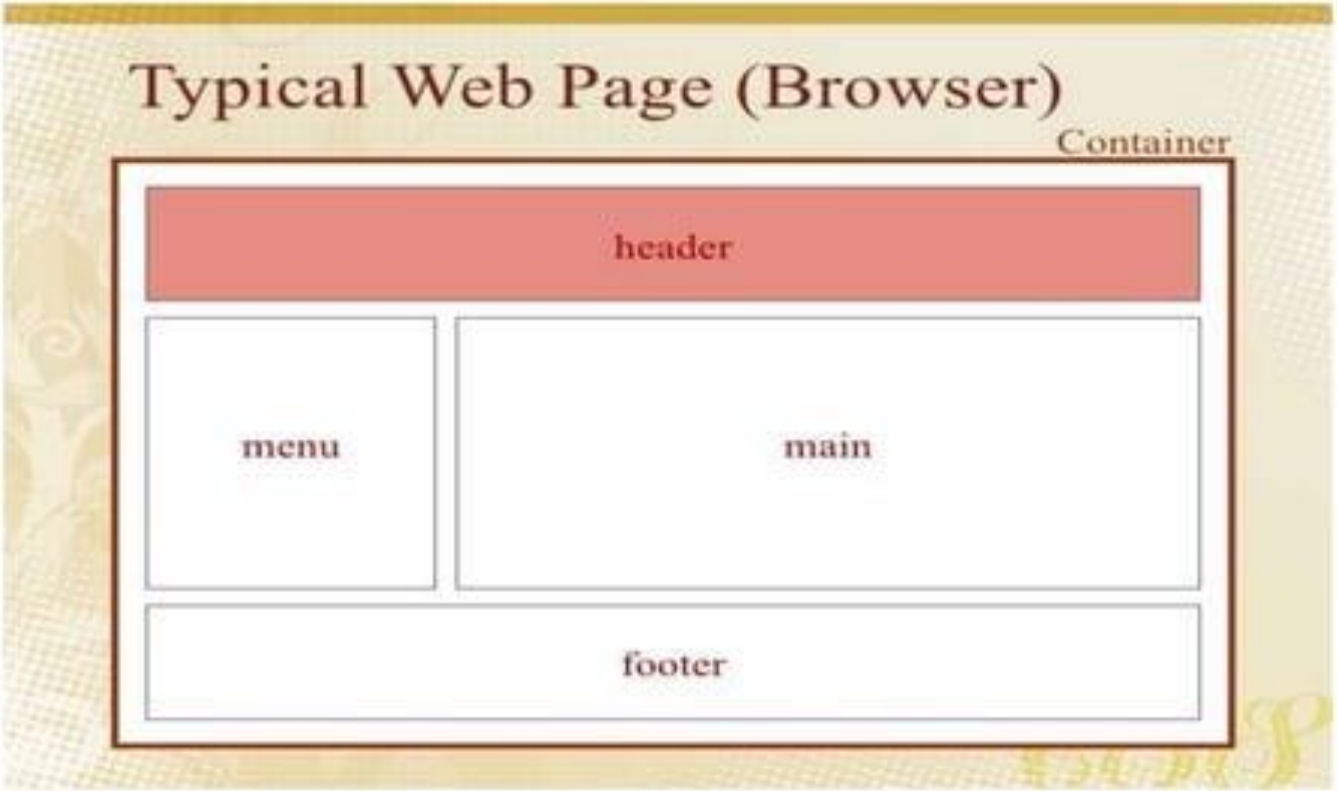
Properties and Values tell an HTML elements how to display.

```
 body
{
background : purple ;
color : green ;
}
```

COMMENT IN CSS:-

- *Explain the purpose of the coding.*
- *Help others read and understand the code.*
- *Server as a reminder to you for what it all means.*
- *Starts with /* and ends with */.*

TYPICAL WEB PAGE:-



TYPICAL WEB PAGE (HTML)

Typical HTML Web page is made up of containers(boxes) or DIV's. Each DIV is assigned an ID or a class.

```
<div id =" container ">
<div id=" header "> Insert Tittle </div>
<div id=" main "> content
<div id=" menu "> content </div>
</div>
<div id=" footer "> content </div>
</div>
```

TYPICAL WEB PAGE (CSS)

The CSS file users t he same DIV / I D / C LASS names as the HTML and uses them to style the elements.

```
#container { property : value ; }
#menu { property : value ; }
#main { property : value ; }
#footer { property : value ; }
```

IDS AND CLASSES:-

. IDs (#) are unique and can only be used once on the page.

. Classes (.) can be used as many times needed.

HTML codes:-

```
<h1 id =" main heading"> Names</h1>
<p class ="name">xyz</p>
```

CSS codes:-

```
#main heading { color : green}
.name { color :red}
```

CSS BOX PROPERTIES:-

- *Background-color*
- *Width*
- *Padding*
- *Margin*
- *Border-width*
- *Border color*
- *Border-style*
- *Background-image*

BACKGROUND COLOR:-

The background-color property specifies the background color of an element.

Example

```
body {
    background-color: light blue;
}
```

BACKGROUND IMAGE

The background-image property specifies an image to use as the background of an element.

By default, the image is repeated so it covers the entire element.

Example:-

```
body {
    background-image: url("paper.gif");
}
```

CSS BORDER STYLE:-

The **border-style** property specifies what kind of border to display.

The following values are allowed:

-dotted - Defines a dotted border

-dashed - Defines a dashed border

-solid - Defines a solid border

-double - Defines a double border

-none - Defines no border

-hidden - Defines a hidden border

CSS BORDER COLOR:-

The **border-color** property is used to set the color of the four borders.

The color can be set by:

name - specify a color name, like "red"

Hex - specify a hex value, like "#ff0000"

Example

```
p.one {
        border-style: solid;
        border-color: red;
    }
```

CSS HEIGHT AND WIDTH:-

The height and width properties are used to set the height and width of an element.

The height and width can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in length values, like px, cm, etc., or in percent (%) of the containing block.

Example:-

```
div {
    width: 500px;
    height: 100px;
    border: 3px solid #73AD21;
}
```

CSS MARGIN:-

The CSS margin properties set the size of the white space OUTSIDE the border.

CSS has properties for specifying the margin for each side of an element:

-margin-top

-margin-right

-margin-bottom

-margin-left

CSS PADDING:-

The CSS padding properties define the white space between the element content and the element border.

The padding clears an area around the content (inside the border) of an element.

CSS has properties for specifying the padding for each side of an element:

-padding-top

-padding-right

-padding-bottom

-padding-left

CSS FONTS:-

The CSS font properties define the font family, boldness, size, and the style of a text.

(a) CSS Font Families:-

In CSS, there are two types of font family names:

generic family *- a group of font families with a similar look (like "Serif" or "Monospace")*

font family *- a specific font family (like "Times New Roman" or "Arial")*

(b) Font Family:-

The font family of a text is set with the font-family property.

Example

```
p {
    font-family: "Times New Roman", Times, serif;
}
```

(c) Font Style:-

The font-style property is mostly used to specify italic text.

This property has three values:

normal - The text is shown normally

italic - The text is shown in italics

Example:-

```
p.normal {
    font-style: normal;
}

p.italic {
    font-style: italic;
}
```

(d) Font Size:-

The font-size property sets the size of the text. Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs.

ADVANTAGES OF CSS:-

-Easier to maintain and update.

-Greater consistency in design.

-More formatting options.

-Lightweight code.

-Faster download times.

-Search engine optimization benefits.

-Ease of presenting different styles to different viewers.

-Greater accessibility.

Thank you

FLASK MICRO FRAME WORK

- **What is Web Framework?**
- Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.
- **WSGI**
- Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications

FLASK ISTALLATION

- **virtualenv** is a virtual Python environment builder. It helps a user to create multiple Python environments side-by-side. Thereby, it can avoid compatibility issues between the different versions of the libraries.
- pip install virtualenv
- **We are now ready to install Flask in this environment.**
- pip install Flask

SAY FLASK TO HELLOW

- In order to test **Flask** installation, type the following code in the editor as **Hello.py**
- Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.
- Flask constructor takes the name of **current module (__name__)** as argument.
- The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
   return 'Hello World’

if __name__ == '__main__':
   app.run()
```

PARAMETERS AND DESCRIPTION

1. **host**
   Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to ‘0.0.0.0’ to have server available externally

2. **port**
   Defaults to 5000

3. **debug**
   Defaults to false. If set to true, provides a debug information

4. **options**
   To be forwarded to underlying Werkzeug server.

Debug mode

- A **Flask** application is started by calling the **run()** method. However, while the application is under development, it should be restarted manually for each change in the code. To avoid this inconvenience, enable **debug support**. The server will then reload itself if the code changes. It will also provide a useful debugger to track the errors if any, in the application.
- The **Debug** mode is enabled by setting the **debug** property of the **application** object to **True** before running or passing the debug parameter to the **run()** method.

```
app.debug = True
app.run()
app.run(debug = True)
```

Flask – Routing

- Modern web frameworks use the routing technique to help a user remember application URLs. It is useful to access the desired page directly without having to navigate from the home page.

```
@app.route('/hello')
def hello_world():
   return 'hello world'
```

FLASK-HTTP METHODS

	Methods & Description
1	**GET** Sends data in unencrypted form to the server. Most common method.
2	**HEAD** Same as GET, but without response body
3	**POST** Used to send HTML form data to server. Data received by POST method is not cached by server.
4	**PUT** Replaces all current representations of the target resource with the uploaded content.
5	**DELETE** Removes all current representations of the target resource given by a URL

CODE FOR GET AND POST

```
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<name>')
def success(name):
   return 'welcome %s' % name

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST':
      user = request.form['nm']
      return redirect(url_for('success',name = user))
   else:
      user = request.args.get('nm')
      return redirect(url_for('success',name = user))

if __name__ == '__main__':
   app.run(debug = True)
```

```
<html>
   <body>
      <form action = "http://localhost:5000/login" method = "post">
         <p>Enter Name:</p>
         <p><input type = "text" name = "nm" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

FLASK TEMPLATES

- It is possible to return the output of a function bound to a certain URL in the form of HTML. For instance, in the following script, **hello()** function will render **'Hello World'** with **<h1>** tag attached to it.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
   return '<html><body><h1>Hello World</h1></body></html>'

if __name__ == '__main__':
   app.run(debug = True)
```

RNEDERING HTML

- However, generating HTML content from Python code is cumbersome, especially when variable data and Python language elements like conditionals or loops need to be put. This would require frequent escaping from HTML.
- This is where one can take advantage of **Jinja2** template engine, on which Flask is based. Instead of returning hardcode HTML from the function, a HTML file can be rendered by the **render_template()** function.

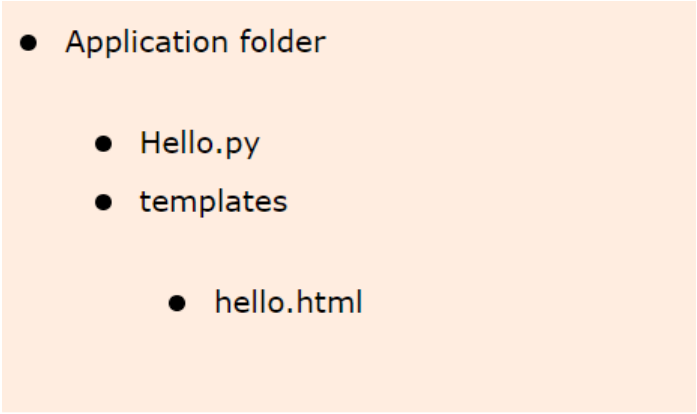
```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
   return render_template('hello.html')

if __name__ == '__main__':
   app.run(debug = True)
```

STRUCTURE OF FINDING TEMPLATES

- Flask will try to find the HTML file in the templates folder, in the same folder in which this script is present.



FLASK STATIC FILES

- A web application often requires a static file such as a **javascript** file or a **CSS** file supporting the display of a web page. Usually, the web server is configured to serve them for you, but during the development, these files are served from *static* folder in your package or next to your module and it will be available at ***/static*** on the application.

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def index():
   return render_template("index.html")

if __name__ == '__main__':
   app.run(debug = True)
```

```
function sayHello() {
   alert("Hello World")
}
```

```
<html>
   <head>
      <script type = "text/javascript"
         src = "{{ url_for('static', filename = 'hello.js') }}" ></script
   </head>

   <body>
      <input type = "button" onclick = "sayHello()" value = "Say Hello" /
   </body>
</html>
```