A Quick Tour of Python

Write/Edit

Run
with some input

OK?

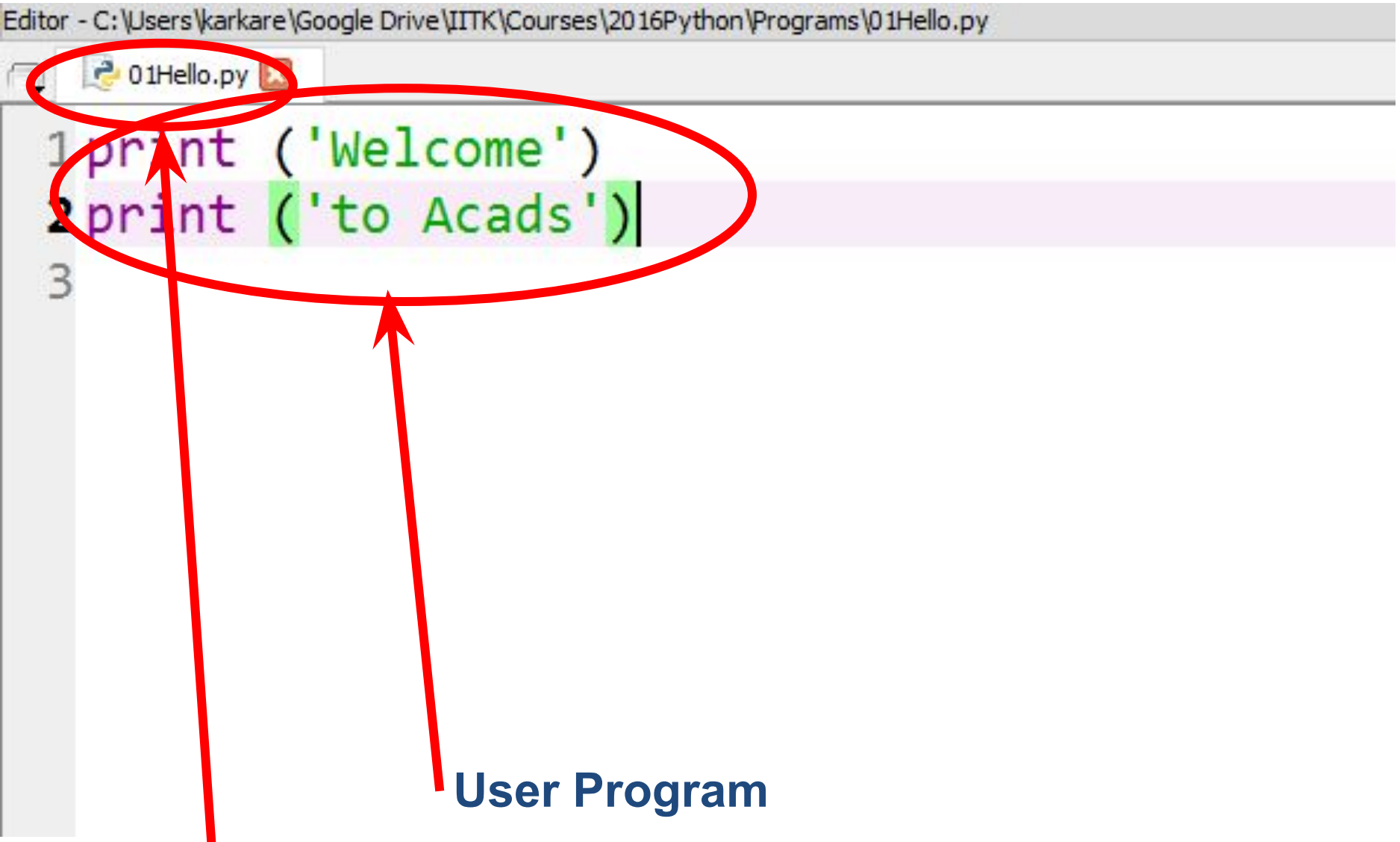
NO

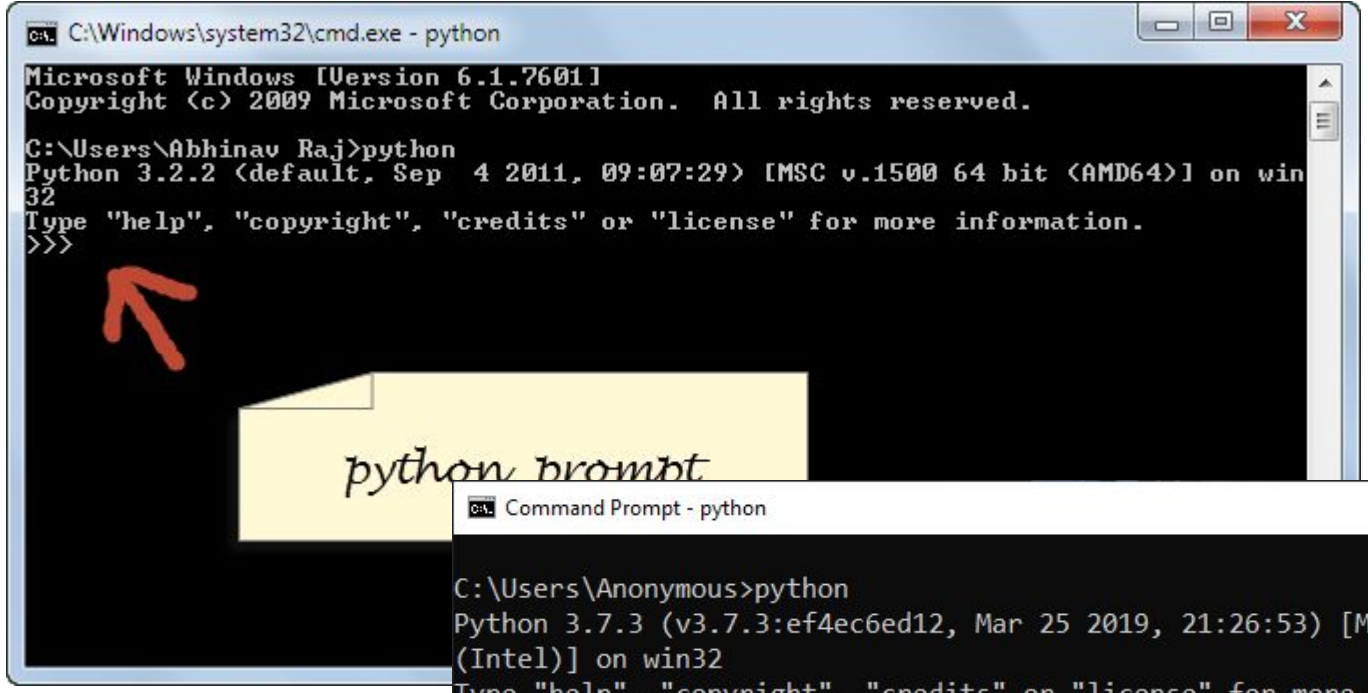
YES

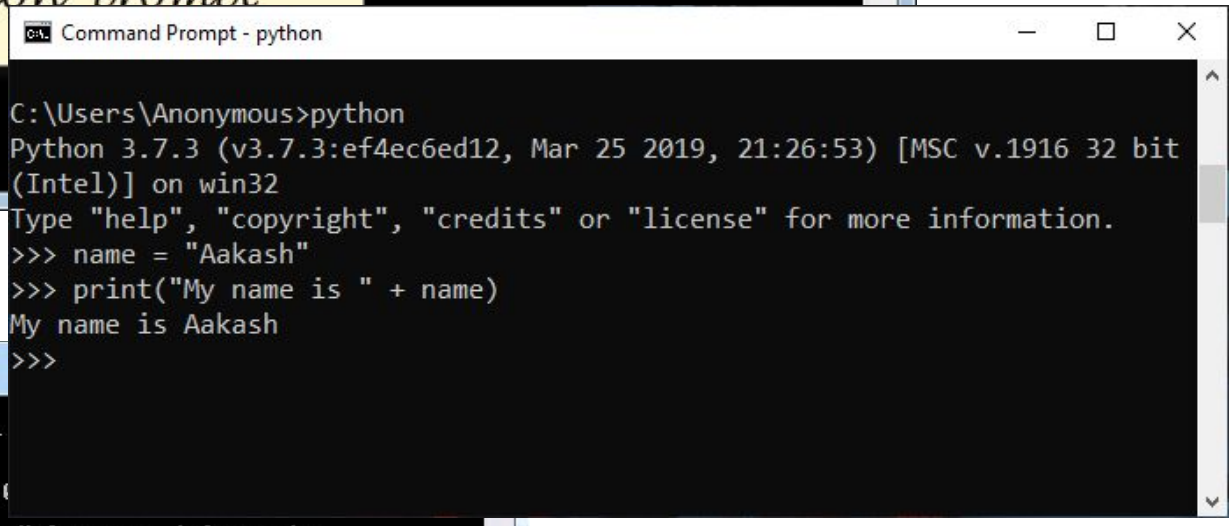
NO

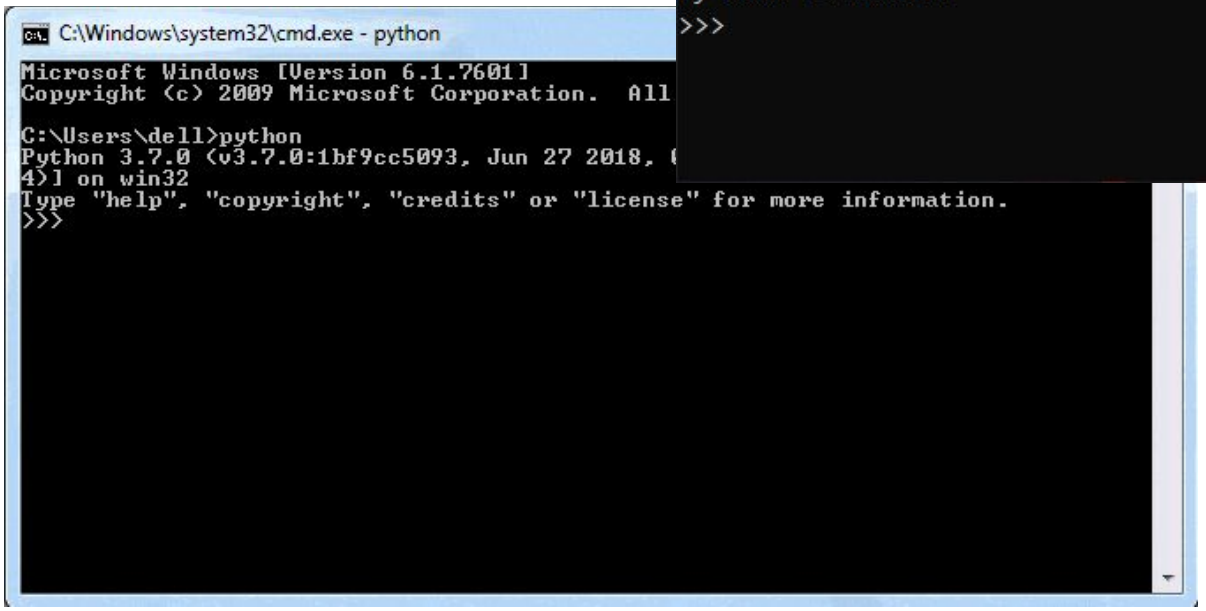
More
Inputs?

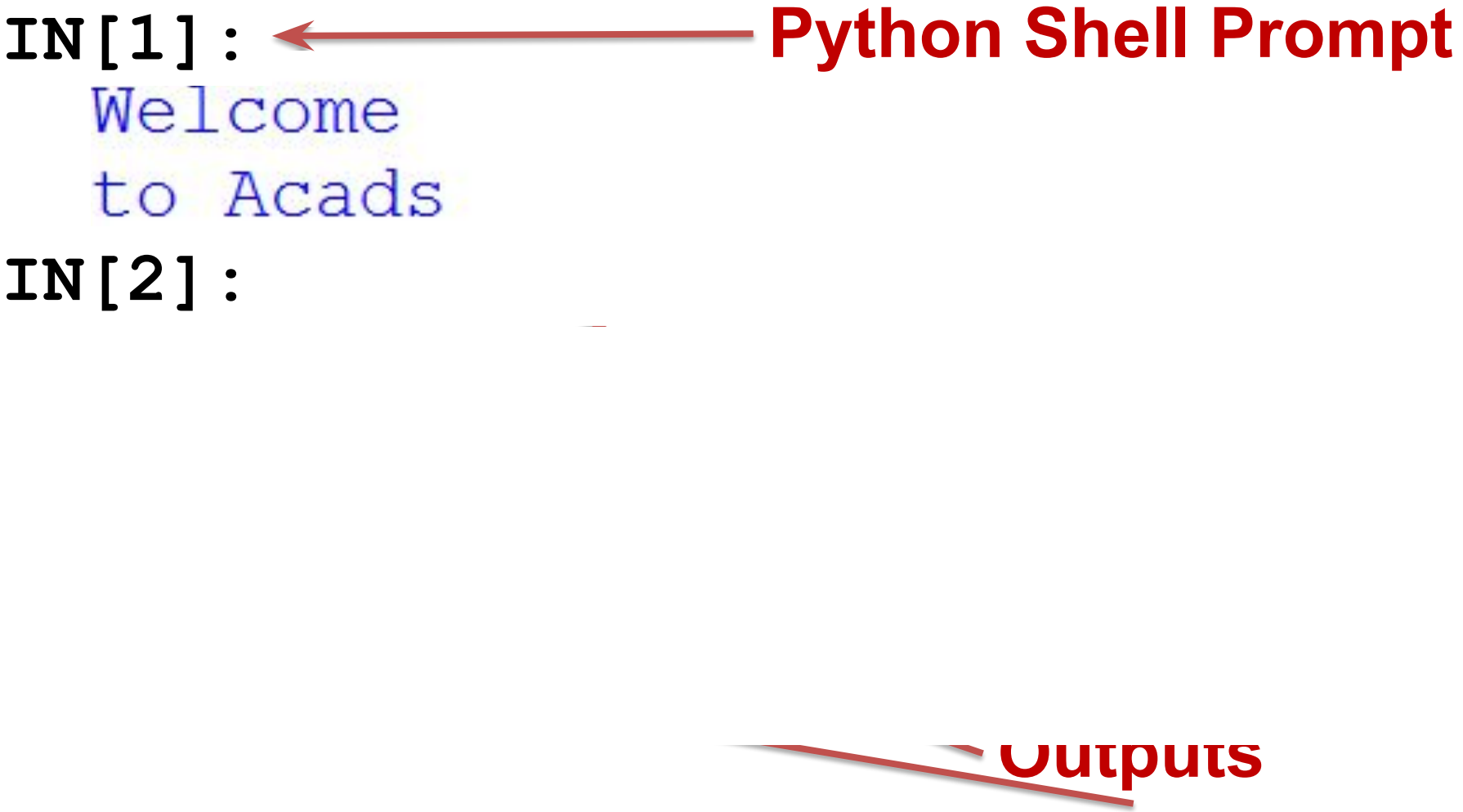
YES


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


C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Abhinav Raj>python
Python 3.2.2 (default, Sep 4 2011, 09:07:29) [MSC v.1500 64 bit (AMD64)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
python prompt


Command Prompt - python
C:\Users\Anonymous>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> name = "Aakash"
>>> print("My name is " + name)
My name is Aakash
>>>


C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All
C:\Users\dell>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018,
4)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>



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’

Leading zeros in non-zero integers are not allowed in Python, e.g. 000123 is invalid number and 0000 becomes 0.

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
- The int() function converts a string or float to int.

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'
```

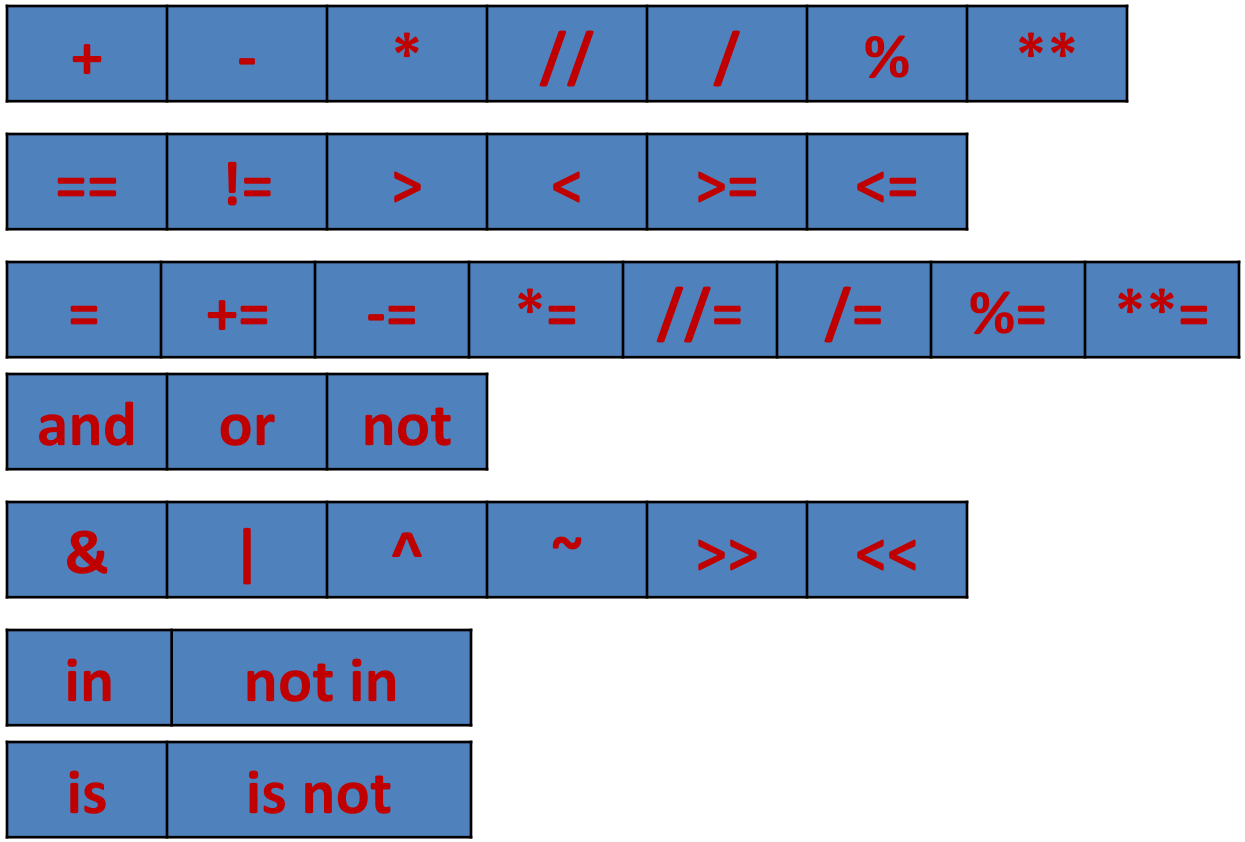
Type Conversion and Input

```
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
- Bitwise
- Membership
- Identity



Variables

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

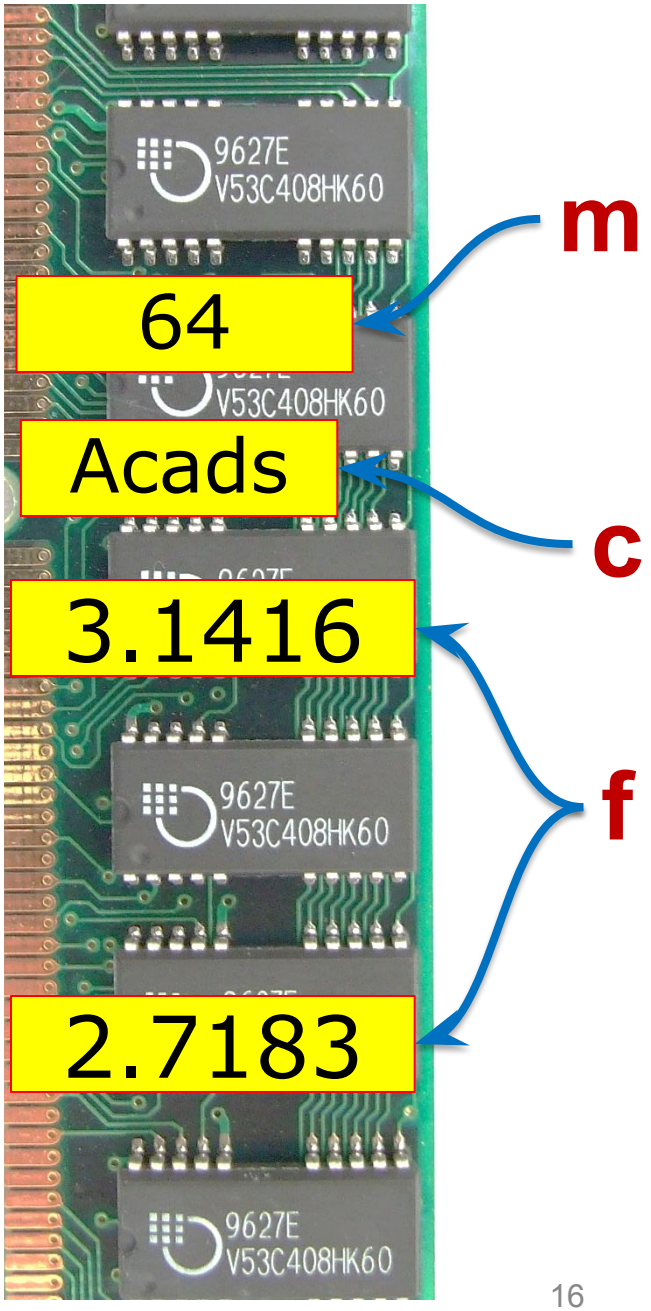
m = 64;

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

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 ('e min')
```

skipped

Run the program

6

10

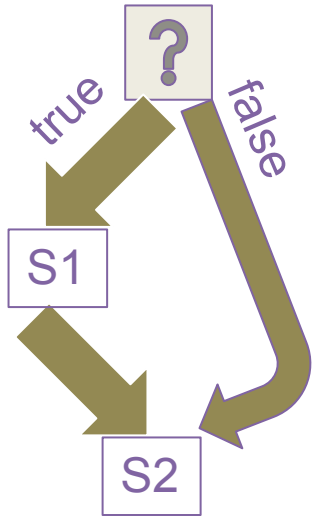
Output

6

if statement (no else!)

- General form of the if statement

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

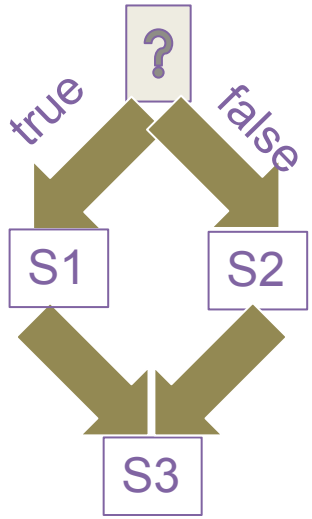


- Execution of if statement
  - First the expression is evaluated.
  - If it evaluates to a **true** value, then S1 is executed and then control moves to the S2.
  - If expression evaluates to **false**, then control moves to the S2 directly.

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

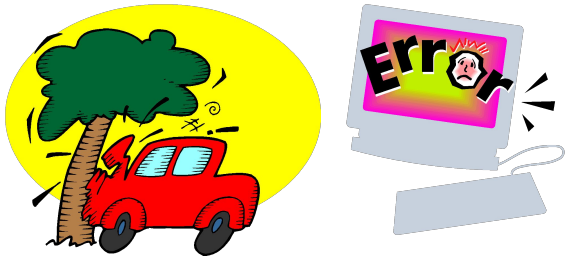
- if-else, nested if's, elif.
- Multiple ways to solve a problem
  - issues of readability, maintainability
  - and efficiency

Class Quiz

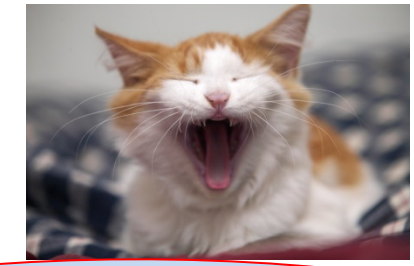
- What is the value of expression:

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

a) Run time crash/error

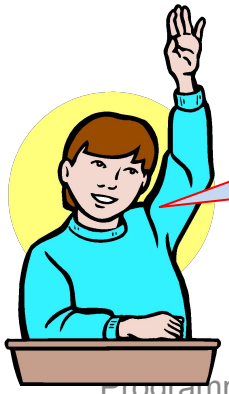


b) I don't know / I don't care



c) False

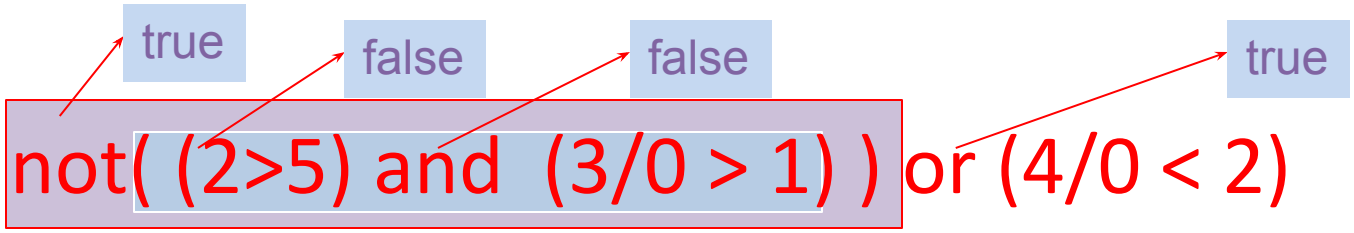
d) True



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

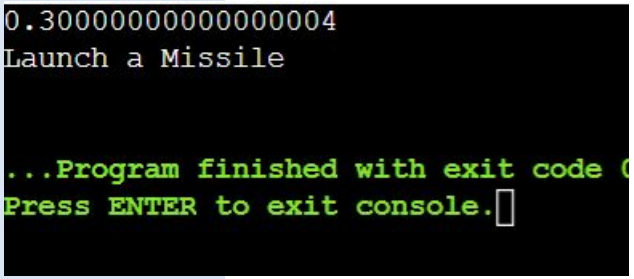
3 Factors for Expr Evaluation

- **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
print(y);
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

  0.0001100110011001100110011001100110011001100110011001100110

- Equivalent to decimal value

  0.1000000000000000055511151231257827021181583404541015625

- Approximation is similar to decimal approximation 1/3 = 0.33333333...
- 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

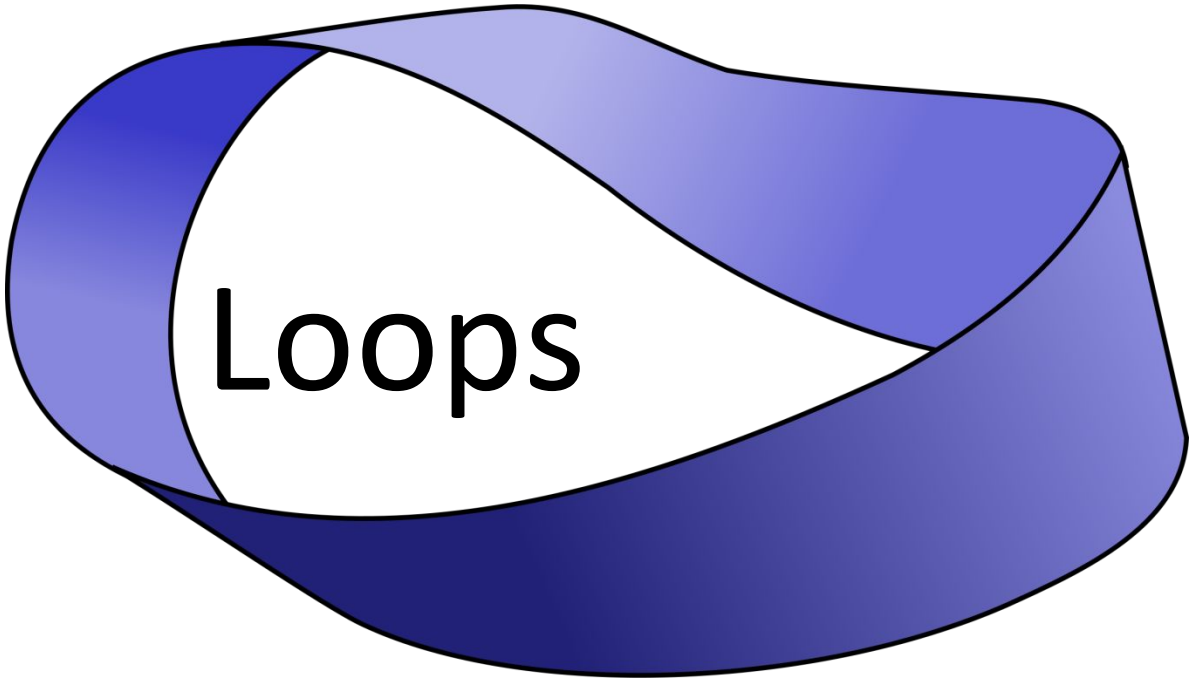
  x == y

  use

  abs(x-y) <= epsilon

  where epsilon is a suitably chosen small value

Programming using Python

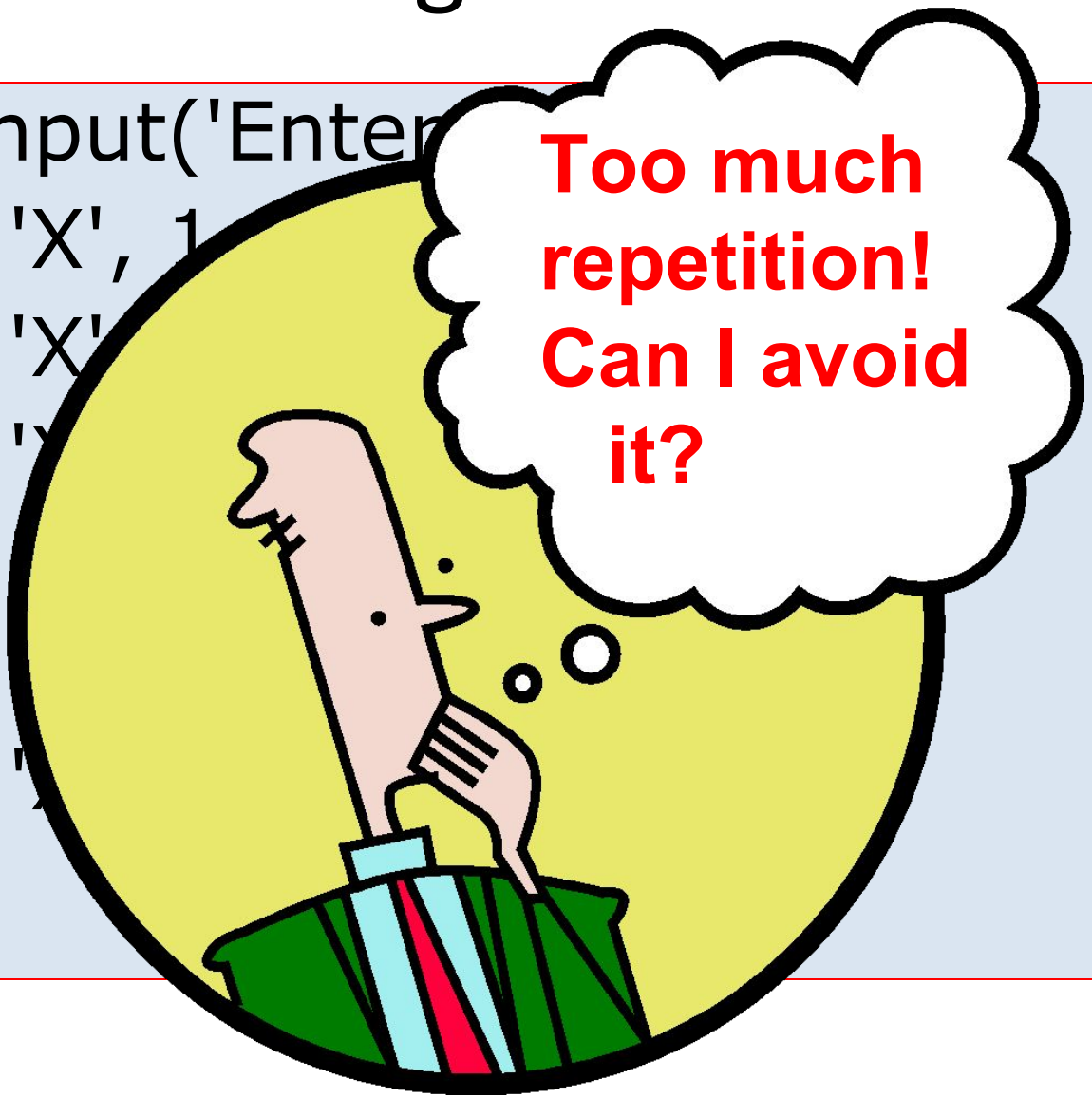


Printing Multiplication Table

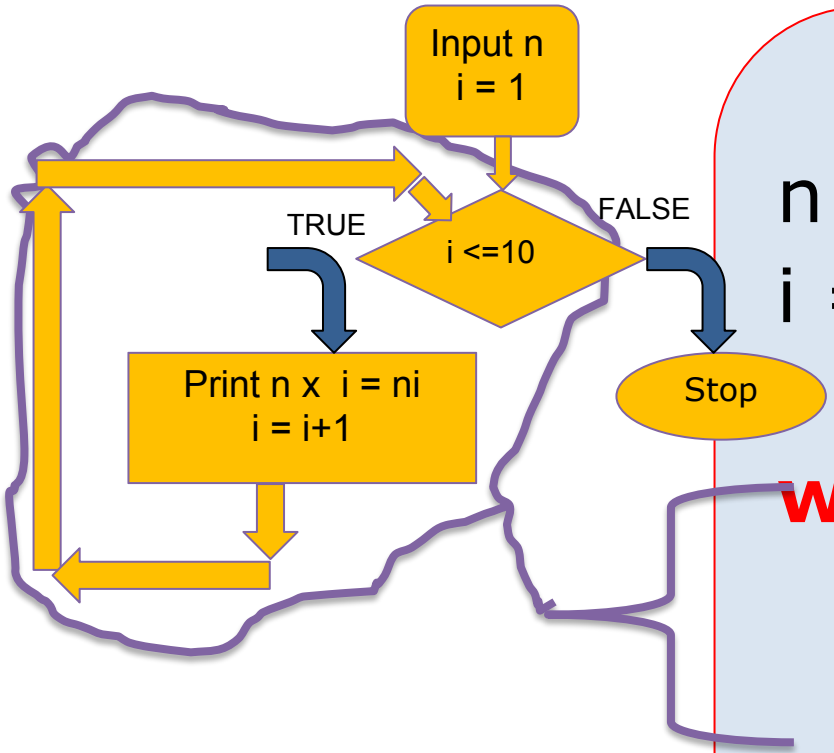
5	X	1	=	5
5	X	2	=	10
5	X	3	=	15
5	X	4	=	20
5	X	5	=	25
5	X	6	=	30
5	X	7	=	35
5	X	8	=	40
5	X	9	=	45
5	X	10	=	50

Program…

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



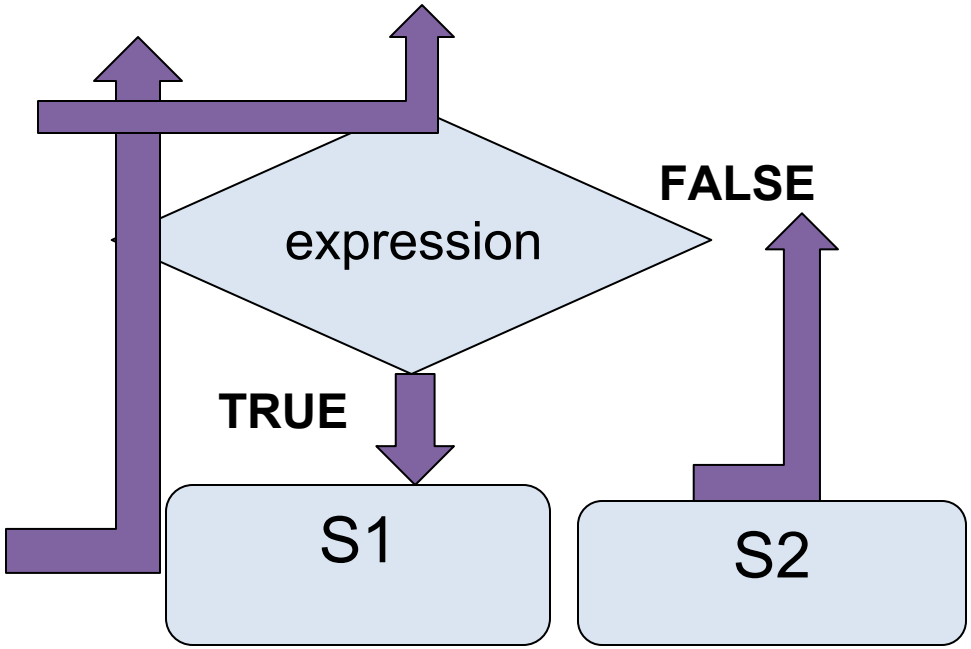
Printing Multiplication Table




```
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
```

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

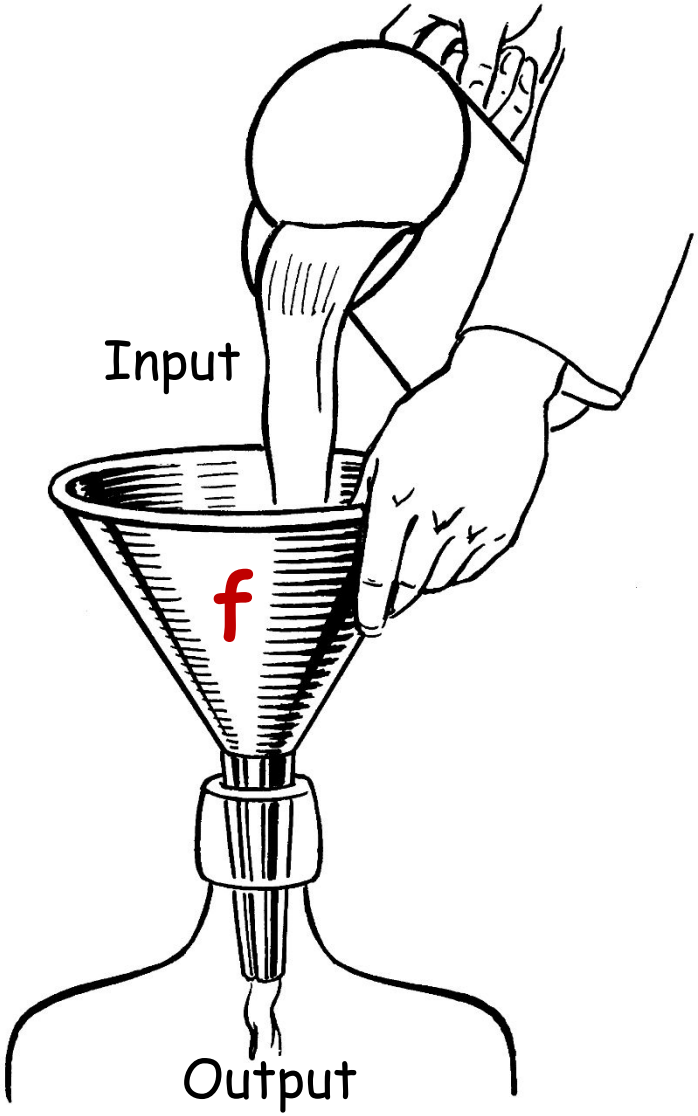
```
# print all odd numbers < 10
i = 1
while i <= 10:
    if i%2==0: # even
        continue
    print (i, end=' ')
    i = i+1
```

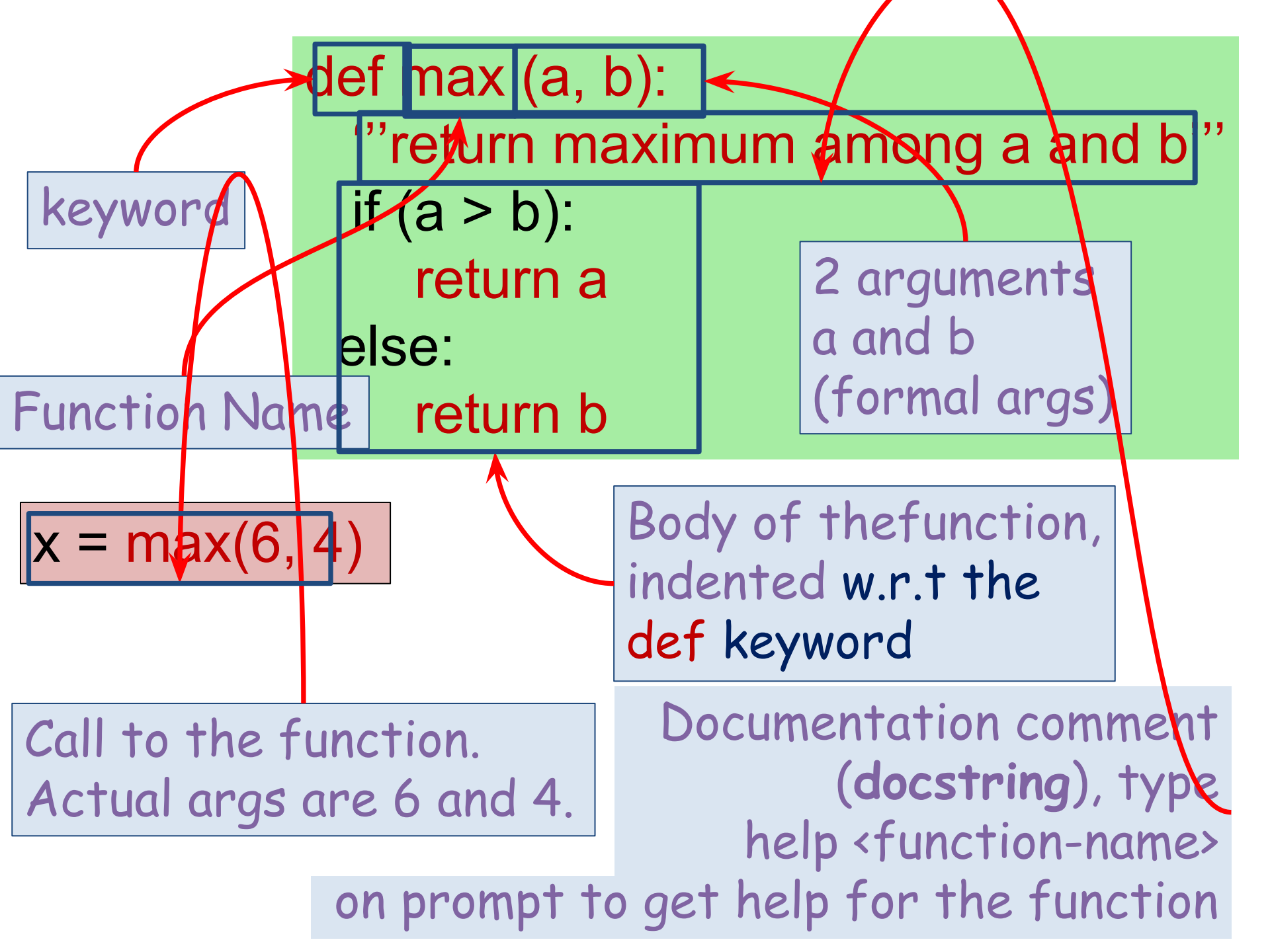
i is not incremented when even number encountered. Infinite loop!!

Programming using Python

f(unctions)

Parts of a function



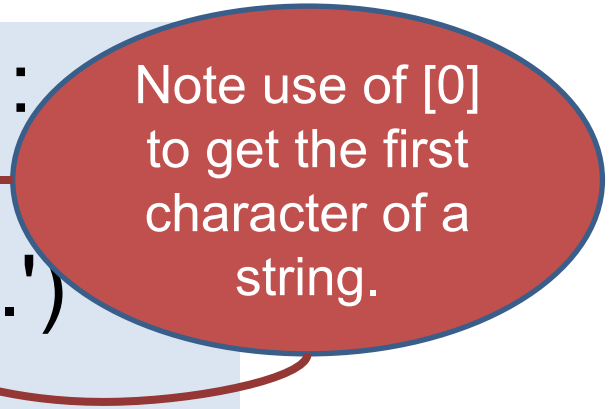

def max(a, b):
"""return maximum among a and b"""
if (a > b):
return a
else:
return b
keyword
Function Name
2 arguments
a and b
(formal args)
x = max(6, 4)
Body of thefunction,
indented w.r.t the
def keyword
Call to the function.
Actual args are 6 and 4.
Documentation comment
(docstring), type
help 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

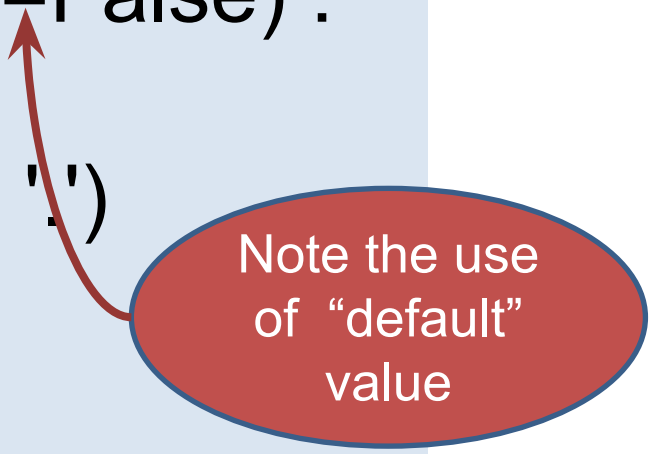
```
def printName(first, last, initials) :
    if initials:
        print (first[0] + '. ' + last[0] + '.')
    else:
        print (first, last)
```



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

Default Values

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



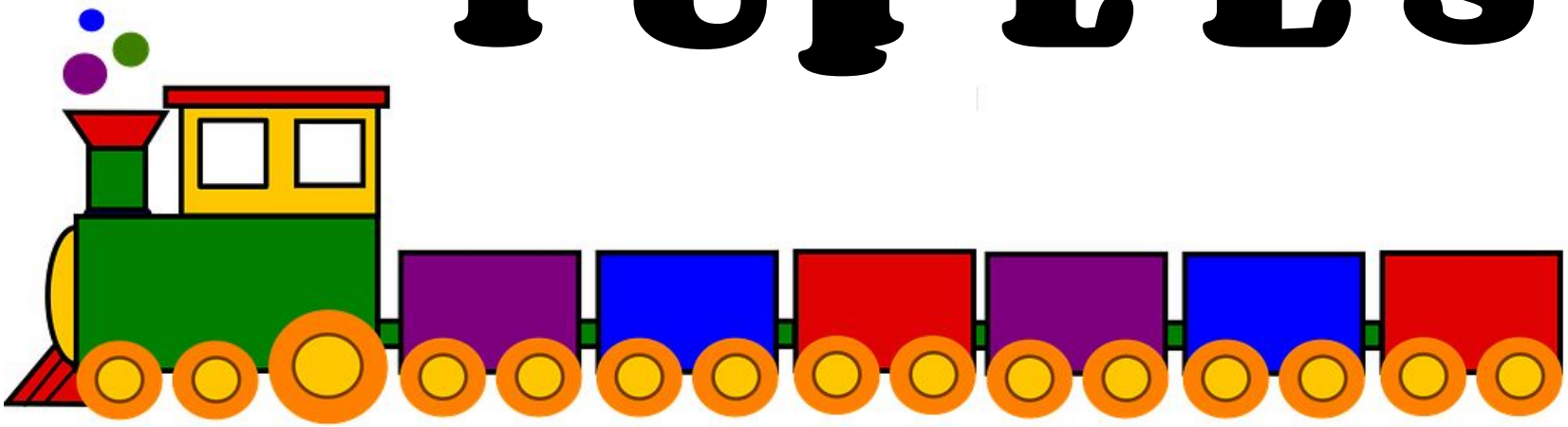
Call	Output
printName('Acads', 'Institute')	Acads Institute

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

Programming with Python

STRINGS
TUPLES



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 (‘ &) n t “ ”)

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

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	

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

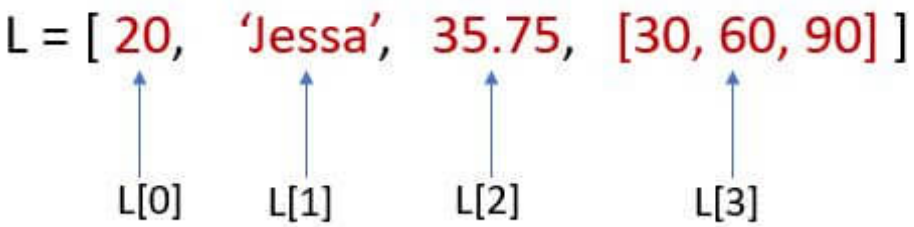
```
>>> student
('Prasanna', 34, ('Python', 'Amey', 101))
>>> name,roll,regdcourse=student
>>> name
```

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'>
```

List in Python



- ✓ **Ordered**: Maintain the order of the data insertion.
- ✓ **Changeable**: List is mutable and we can modify items.
- ✓ **Heterogeneous**: List can contain data of different types
- ✓ **Contains duplicate**: Allows duplicates data

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.

Arrays

- An array is a vector containing homogeneous elements i.e. belonging to the same data type.
- Elements are allocated with contiguous memory locations. Typically the size of an array is fixed.

Creating Python Arrays

To create an array of numeric values, we need to import the `array` module. For example:

```
import array as arr
a = arr.array('d', [1.1, 3.5, 4.5])
print(a)
```

Output

```
array('d', [1.1, 3.5, 4.5])
```

Arrays

Creating Python Arrays

To create an array of numeric values, we need to import the array module. For example:

```
import array as arr
a = arr.array('d', [1.1, 3.5, 4.5])
print(a)
```

Output

```
array('d', [1.1, 3.5, 4.5])
```

Code	C Type	Python Type	Min bytes
b	signed char	int	1
B	unsigned char	int	1
u	Py_UNICODE	Unicode	2
h	signed short	int	2
H	unsigned short	int	2
i	signed int	int	2
I	unsigned int	int	2
l	signed long	int	4
L	unsigned long	int	4
f	float	float	4
d	double	float	8

Accessing Python Array Elements

We use indices to access elements of an array:

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])

print("First element:", a[0])
print("Second element:", a[1])
print("Last element:", a[-1])
```

Output

```
First element: 2
Second element: 4
Last element: 8
```

```
import array as arr

numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
numbers_array = arr.array('i', numbers_list)

print(numbers_array[2:5]) # 3rd to 5th
print(numbers_array[:-5]) # beginning to 4th
print(numbers_array[5:])  # 6th to end
print(numbers_array[:])   # beginning to end
```

Output

```
array('i', [62, 5, 42])
array('i', [2, 5, 62])
array('i', [52, 48, 5])
array('i', [2, 5, 62, 5, 42, 52, 48, 5])
```

Slicing Python Arrays

We can access a range of items in an array by using the slicing operator `:`.

Changing and Adding Elements

Arrays are mutable; their elements can be changed in a similar way as lists.

```
import array as arr

numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

# changing first element
numbers[0] = 0
print(numbers)     # Output: array('i', [0, 2, 3, 5, 7, 10])

# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)     # Output: array('i', [0, 2, 4, 6, 8, 10])
```

Output
array('i', [0, 2, 3, 5, 7, 10])
array('i', [0, 2, 4, 6, 8, 10])

We can add one item to the array using the append() method, or add several items using the extend() method.

```
import array as arr

numbers = arr.array('i', [1, 2, 3])

numbers.append(4)
print(numbers)     # Output: array('i', [1, 2, 3, 4])

# extend() appends iterable to the end of the array
numbers.extend([5, 6, 7])
print(numbers)     # Output: array('i', [1, 2, 3, 4, 5, 6, 7])
```

Output

```
array('i', [1, 2, 3, 4])
array('i', [1, 2, 3, 4, 5, 6, 7])
```

We can also concatenate two arrays using `+` operator.

```
import array as arr

odd = arr.array('i', [1, 3, 5])
even = arr.array('i', [2, 4, 6])

numbers = arr.array('i')   # creating empty array of integer
numbers = odd + even

print(numbers)
```

Output

```
array('i', [1, 3, 5, 2, 4, 6])
```

Removing Python Array Elements

We can delete one or more items from an array using Python's del

```
import array as arr

number = arr.array('i', [1, 2, 3, 3, 4])

del number[2]  # removing third element
print(number)  # Output: array('i', [1, 2, 3, 4])

del number  # deleting entire array
print(number)  # Error: array is not defined
```

Output

```
array('i', [1, 2, 3, 4])
Traceback (most recent call last):
  File "<string>", line 9, in <module>
    print(number)  # Error: array is not defined
NameError: name 'number' is not defined
```

```
import array as arr

numbers = arr.array('i', [10, 11, 12, 12, 13])

numbers.remove(12)
print(numbers)   # Output: array('i', [10, 11, 12, 13])

print(numbers.pop(2))   # Output: 12
print(numbers)   # Output: array('i', [10, 11, 13])
```

Output

```
array('i', [10, 11, 12, 13])
12
array('i', [10, 11, 13])
```

We can use the remove() method to remove the given item, and pop() method to remove an item at the given index.

Examples

Removal all characters from a string except integers

Given:
str1 = 'I am 25 years and 10 months old’

Expected Output: 2510

```
str1 = 'I am 25 years and 10 months old’
print("Original string is", str1) # Retain Numbers in
String
# Using list comprehension + join() + isdigit()
res = "".join([item for item in str1 if item.isdigit()])
print(res)
```

For string : isinstance(var, str)
Fr characters: ischar()

Write a program to check if two strings are balanced.
For example, strings s1 and s2 are balanced if all the characters in the s1 are present in s2. The character's position doesn't matter.

Case 1:
s1 = "Yn"
s2 = "PYnative"
Expected Output: True

Case 2:
s1 = "Ynf"
s2 = "PYnative"
Expected Output:False

```
def string_balance_test(s1, s2):
    flag = True
    for char in s1:
        if char in s2:
            continue
        else:
            flag = False
    return flag


s1 = "Yn"
s2 = "PYnative"
flag = string_balance_test(s1, s2)
print("s1 and s2 are balanced:", flag)

s1 = "Ynf"
s2 = "PYnative"
flag = string_balance_test(s1, s2)
print("s1 and s2 are balanced:", flag)
```

Given two strings, s1 and s2, write a program to return a new string made of s1 and s2's first, middle, and last characters.

Given:
s1 = "America" s2 = "Japan"

Expected Output: AJrpan

```
def mix_string(s1, s2):
    # get first character from both string
    first_char = s1[0] + s2[0]

    # get middle character from both string
    middle_char = s1[int(len(s1) / 2):int(len(s1) / 2) + 1] + s2[int(len(s2) / 2

    # get last character from both string
    last_char = s1[len(s1) - 1] + s2[len(s2) - 1]

    # add all
    res = first_char + middle_char + last_char
    print("Mix String is ", res)

s1 = "America"
s2 = "Japan"
mix_string(s1, s2)
```

Python Program to Solve Quadratic Equation

```
# Solve the quadratic equation ax**2 + bx + c = 0

# import complex math module
import cmath

a = 1
b = 5
c = 6

# calculate the discriminant
d = (b**2) - (4*a*c)

# find two solutions
sol1 = (-b-cmath.sqrt(d))/(2*a)
sol2 = (-b+cmath.sqrt(d))/(2*a)

print('The solution are {0} and {1}'.format(sol1,sol2))
```

Output

```
Enter a: 1
Enter b: 5
Enter c: 6
The solutions are (-3+0j) and (-2+0j)
```

Python Program to Display the multiplication TablE

```
# Multiplication table (from 1 to 10) in Python

num = 12

# To take input from the user
# num = int(input("Display multiplication table of? "))

# Iterate 10 times from i = 1 to 10
for i in range(1, 11):
   print(num, 'x', i, '=', num*i)
```

Output

```
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
12 x 7 = 84
12 x 8 = 96
12 x 9 = 108
12 x 10 = 120
```

Python Program to Make a Simple Calculator

```
# This function adds two numbers
def add(x, y):
    return x + y

# This function subtracts two numbers
def subtract(x, y):
    return x - y

# This function multiplies two numbers
def multiply(x, y):
    return x * y

# This function divides two numbers
def divide(x, y):
    return x / y


print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
```

```
while True:
    # take input from the user
    choice = input("Enter choice(1/2/3/4): ")

    # check if choice is one of the four options
    if choice in ('1', '2', '3', '4'):
        try:
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
            print("Invalid input. Please enter a number.")
            continue

        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))

        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))

        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))

        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))

        # check if user wants another calculation
        # break the while loop if answer is no
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
          break
else:
    print("Invalid Input")
```

Output

```
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 3
Enter first number: 15
Enter second number: 14
15.0 * 14.0 = 210.0
Let's do next calculation? (yes/no): no
```

Python Program to Shuffle Deck of Cards

```
# Python program to shuffle a deck of card

# importing modules
import itertools, random

# make a deck of cards
deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))

# shuffle the cards
random.shuffle(deck)

# draw five cards
print("You got:")
for i in range(5):
   print(deck[i][0], "of", deck[i][1])
```

Output

```
You got:
5 of Heart
1 of Heart
8 of Spade
12 of Spade
4 of Spade
```

Python Program to Display Calendar

```
# Program to display calendar of the given month and year

# importing calendar module
import calendar

yy = 2014  # year
mm = 11    # month

# To take month and year input from the user
# yy = int(input("Enter year: "))
# mm = int(input("Enter month: "))

# display the calendar
print(calendar.month(yy, mm))
```

Output

```
   November 2014
Mo Tu We Th Fr Sa Su
                1  2
3  4  5  6 7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

Python Program to Split a List Into Evenly Sized Chunk

```
import numpy as np

my_list = [1,2,3,4,5,6,7,8,9]
print(np.array_split(my_list, 5))
```

Output

```
[array([1, 2]), array([3, 4]), array([5, 6]), array([7, 8]), array([9])]
```

Python Program to Print Output Without a Newline

Using the end keyword, you can append a string at the end of the print text.

```
# print each statement on a new line
print("Python")
print("is easy to learn.")

# new line
print()

# print both the statements on a single line
print("Python", end=" ")
print("is easy to learn.")
```

Output

```
Python
is easy to learn.

Python is easy to learn.
```

Remove Punctuations From a String

```
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

my_str = "Hello!!!, he said ---and went."

# To take input from the user
# my_str = input("Enter a string: ")

# remove punctuation from the string
no_punct = ""
for char in my_str:
   if char not in punctuations:
       no_punct = no_punct + char

# display the unpunctuated string
print(no_punct)
```

Output

```
Hello he said and went
```

```
my_string = " Python "

print(my_string.strip())
```

Output

```
Python
```

Sort Words in Alphabetic Order

```
# Program to sort alphabetically the words form a string provided by the user

my_str = "Hello this Is an Example With cased letters"

# To take input from the user
#my_str = input("Enter a string: ")

# breakdown the string into a list of words
words = [word.lower() for word in my_str.split()]

# sort the list
words.sort()

# display the sorted words

print("The sorted words are:")
for word in words:
   print(word)
```

Output

```
The sorted words are:
an
cased
example
hello
is
letters
this
with
```

Program to Capitalize the First Character of a String

```
my_string = "programiz is Lit"

print(my_string[0].upper() + my_string[1:])
```

Output

```
Programiz is Lit
```

```
my_string = "programiz is Lit"

cap_string = my_string.capitalize()

print(cap_string)
```

Output

```
Programiz is lit
```

Write a Python program to add 'ing' at the end of a given string (length should be at least 3).
If the given string already ends with 'ing' then add 'ly' instead.
If the string length of the given string is less than 3, leave it unchanged.

```
def add_string(str1):
  length = len(str1)

  if length > 2:
    if str1[-3:] == 'ing':
      str1 += 'ly'
    else:
      str1 += 'ing'

  return str1
print(add_string('ab'))
print(add_string('abc'))
print(add_string('string'))
```

Sample Output:
ab
abcing
stringly

Python program to remove characters that have odd index values in a given string.

```
def odd_values_string(str):
  result = ""
  for i in range(len(str)):
    if i % 2 == 0:
      result = result + str[i]
  return result

print(odd_values_string('abcdef'))
print(odd_values_string('python'))
```

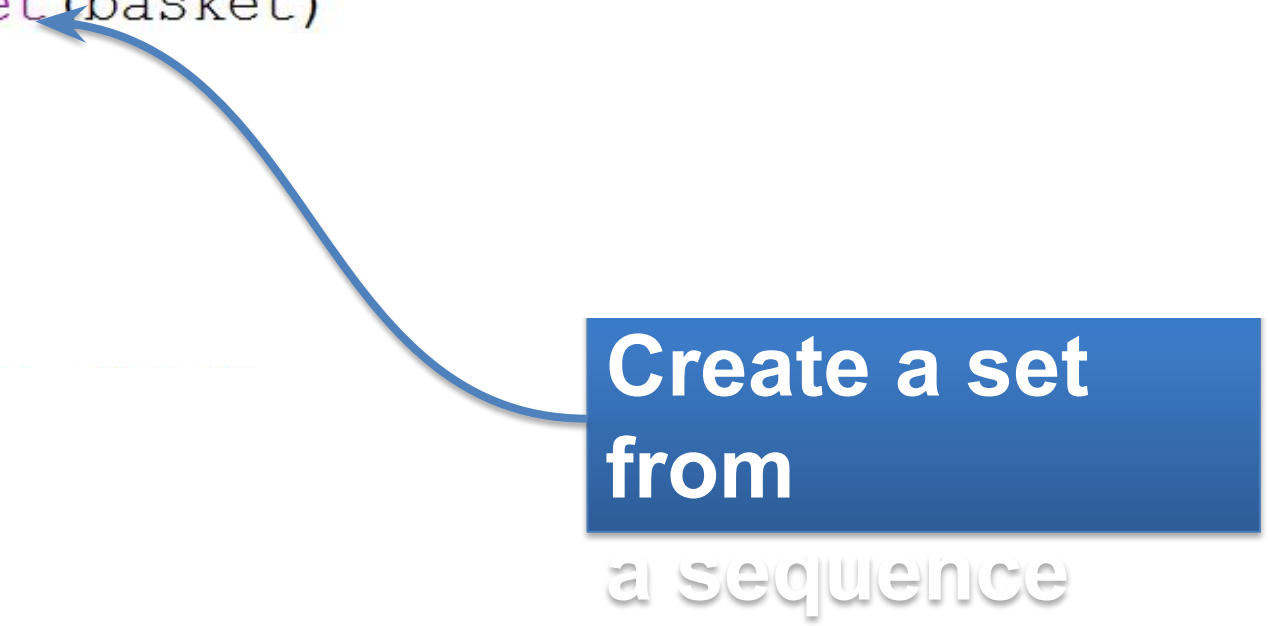
Sample Output:
ace pto

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'}
```