SILVER OAK UNIVERSITY

EDUCATION TO INNOVATION

SILVER OAK COLLEGE OF COMPUTER APPLICATION

SUBJECT :Programming in Python

TOPIC : Intro to Python

- **What is Python?**
- Python is an example of a high-level language; other high-level languages you might have heard of are C++, PHP, and Java.

- Multi-purpose (Web, GUI, Scripting, etc.)
- • Object Oriented
- • Interpreted
- • Strongly typed and Dynamically typed
- • Focus on readability and productivity

- ***History of Python:***
- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherland

- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

- • Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

- • Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

- • Python 1.0 was released in November 1994. In 2000, Python 2.0 was released. Python
- 2.7.11 is the latest edition of Python 2.
- • Meanwhile, Python 3.0 was released in 2008. Python 3 is not backward compatible with Python 2. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules so that "There should be one -- and preferably only one -- obvious way to do it." Python 3.5.1 is the latest version of Python 3

Features of python:

- Python's features include-

- • **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows a student to pick up the language quickly.

- • **Easy-to-read:** Python code is more clearly defined and visible to the eyes.

- • **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.

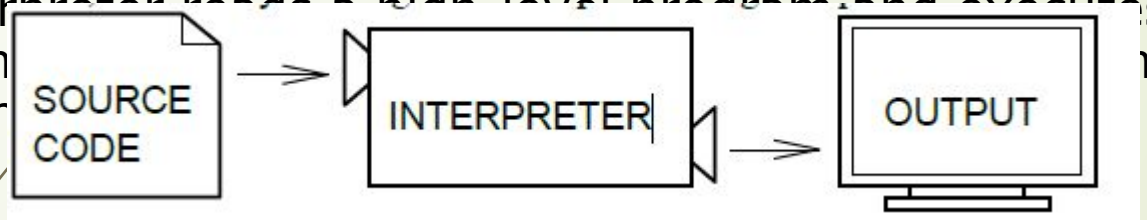
- • **A broad standard library:** Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

- • **Interactive Mode:** Python has support for an interactive mode, which allows interactive testing and debugging of snippets of code.

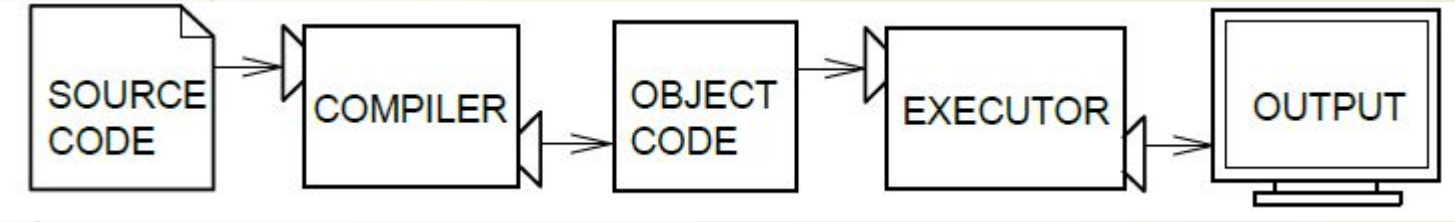
- • **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- • **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- • **Databases:** Python provides interfaces to all major commercial databases.
- • **GUI Programming:** Python supports GUI applications that can be created and ported
- 4 to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- • **Scalable:** Python provides a better structure and support for large programs than shell scripting.

- Apart from the above-mentioned features, Python has a big list of good features. A few are listed below-
- • It supports functional and structured programming methods as well as OOP.
- • It can be used as a scripting language or can be compiled to byte-code for building large applications.
- • It provides very high-level dynamic data types and supports dynamic type checking.
- • It supports automatic garbage collection.
- • It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

- **Interpreter VS Compiler**
- Two kinds of applications process high-level languages into low-level languages: **interpreters** and **compilers**.
- An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and perform



- ---
- A compiler reads the program and translates it into a low-level program, which can then be run.
- In this case, the high-level program is called the **source code**, and the translated program is called the **object code** or the **executable**. Once a program is compiled, you can execute it repeatedly without further translation.



- Many modern languages use both processes. They are first compiled into a lower
- level language, called **byte code**, and then interpreted by a program called **a virtual machine.** Python uses both processes, but because of the way programmers interact with it, it is usually considered an interpreted language
- ----------

- There are two ways to use the Python interpreter: **shell mode and script mode**.
- In shell mode, you type Python statements into the **Python shell** and the interpreter immediately prints the result.

- In this course, we will be using an IDE (Integrated Development Environment) called IDLE. When you first start IDLE it will open an interpreter window.1
-
- The first few lines identify the version of Python being used as well as a few other messages; you can safely ignore the lines about the firewall. Next there is a line identifying the version of IDLE. The last line starts with >>>, which is the **Python prompt**. The interpreter uses the prompt to indicate that it is ready for instructions.
- If we type print 1 + 1 the interpreter will reply 2 and give us another prompt.2
-

- Running Python program:

- There are three different ways to start Python-

- **(1) Interactive Interpreter**
- You can start Python from Unix, DOS, or any other system that provides you a command line interpreter or shell window.
- Enter python the command line.

- **(2) Script from the Command-line**

- **(3) Integrated Development Environment**

- ***What is Debugging ?***
- Programming is a complex process, and because it is done by human beings, programs often contain errors. For whimsical reasons, programming errors are called **bugs** and the process of tracking them down and correcting them is called **debugging**.
- Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors. It is useful to distinguish between them in order to track them down more quickly

---

- ***Syntax errors***
- Python can only execute a program if the program is syntactically correct; otherwise, the process fails and returns an error message. **Syntax** refers to the structure of a program and the rules about that structure. For example, in English, a sentence must begin with a capital letter and end with a period. this sentence contains a **syntax error**.
- So does this one For most readers, a few syntax errors are not a significant problem, which is why we can read the poetry of e. e. cummings without spewing error messages. Python is not so forgiving. If there is a single syntax error anywhere in your program, Python will print an error message and quit, and you will not be able to run your program.
- During the first few weeks of your programming career, you will probably spend a lot of time tracking down syntax errors. As you gain experience, though, you will make fewer syntax errors and find them faster.

- ***Runtime errors***
- The second type of error is a runtime error, so called because the error does not appear until you run the program. These errors are also called **exceptions** because they usually indicate that something exceptional (and bad) has happened. Runtime errors are rare in the simple programs you will see in the first few chapters, so it might be a while before you encounter one.

- ***Semantic errors***
- The third type of error is the **semantic error**. If there is a semantic error in your program, it will run successfully, in the sense that the computer will not generate any error messages, but it will not do the right thing. It will do something else.
- Specifically, it will do what you told it to do.
- The problem is that the program you wrote is not the program you wanted to write. The meaning of the program (its semantics) is wrong. Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

- ***Formal and Natural Languages:***
- **Natural languages** are the languages that people speak, such as English, Spanish, and French. They were not designed by people (although people try to impose some order on them); they evolved naturally.
- **Formal languages** are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols.

- ***Programming languages are formal languages that have been designed to express computations***.

- ----------
- Although formal and natural languages have many features in common—tokens, structure, syntax, and semantics—there are many differences:
- **ambiguity:** Natural languages are full of ambiguity, which people deal with by using contextual clues and other information. Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context.
- **redundancy:** In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose. Formal languages are less redundant and more concise.
- **literalness:** Natural languages are full of idiom and metaphor. If someone says, “The penny dropped”,

- People who grow up speaking a natural language—everyone—often have a hard time adjusting to formal languages. In some ways, the difference between formal and natural language is like the difference between poetry and prose, but more so:
- **Poetry:** Words are used for their sounds as well as for their meaning, and the whole poem together creates an effect or emotional response. Ambiguity is not only common but often deliberate.
- **Prose:** The literal meaning of words is more important, and the structure contributes
- 12 more meaning. Prose is more amenable to analysis than poetry but still often ambiguous.
- **Programs:** The meaning of a computer program is unambiguous and literal, and can be understood entirely by analysis of the tokens and structure.

- ***The Difference Between Brackets, Braces, and Parentheses:***

- Braces are used for different purposes. If you just want a list to contain some elements and organize them by index numbers (starting from 0), just use the [] and add elements as necessary. {} are special in that you can give custom id's to values like a = {"John": 14}. Now, instead of making a list with ages and remembering whose age is where, you can just access John's age by a["John"].
- ***The [] is called a list and {} is called a dictionary (in Python).***
- Dictionaries are basically a convenient form of list which allow you to access data in a much easier way.
- However, there is a catch to dictionaries. Many times, the data that you put in the dictionary doesn't stay in the same order as before. Hence, when you go through each value one by one, it won't be in the order you expect. There is a special dictionary to get around this, but you have to add this line from collections import OrderedDict and replace {} with OrderedDict(). But, I don't think you will need to worry about that for now.

The Programming Cycle for Python

Write/Edit

Run
with some input

OK?

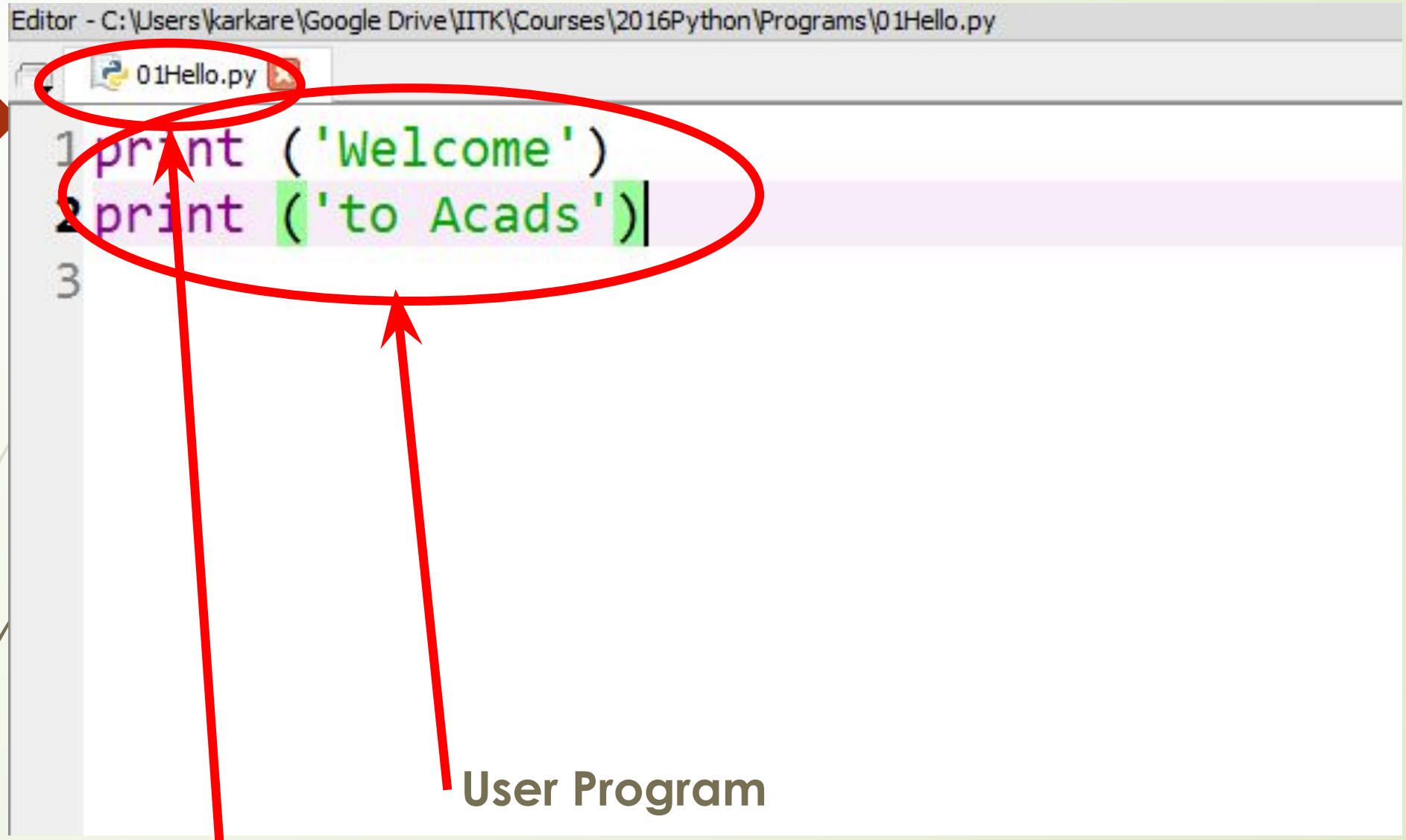
NO

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

Keywords & Identifiers

- **Python Keywords** are some predefined and reserved words in Python that have special meanings. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, or variable name. All the keywords in Python are written in lowercase except True and False. There are 35 keywords in Python 3.11.

Keywords	Description
and	This is a logical operator which returns true if both the operands are true else returns false.
or	This is also a logical operator which returns true if anyone operand is true else returns false.
not	This is again a logical operator it returns True if the operand is false else returns false.
if	This is used to make a conditional statement.
elif	Elif is a condition statement used with an if statement. The elif statement is executed if the previous conditions were not true.
else	Else is used with if and elif conditional statements. The else block is executed if the given condition is not true.
for	This is used to create a loop.
while	This keyword is used to create a while loop.

For more visit this link:
https://www.geeksforgeeks.org/python-keywords-and-identifiers/

Identifiers in Python

Identifier is a user-defined name given to a variable, function, class, module, etc. The identifier is a combination of character digits and an underscore. They are case-sensitive i.e., 'num' and 'Num' and 'NUM' are three different identifiers in python. It is a good programming practice to give meaningful names to identifiers to make the code understandable.

Rules for Naming Python Identifiers

- It cannot be a reserved python keyword.
- It should not contain white space.
- It can be a combination of A-Z, a-z, 0-9, or underscore.
- It should start with an alphabet character or an underscore (_).
- It should not contain any special character other than an underscore (_).

Examples of Python Identifiers

Valid identifiers:

- var1
- _var1
- _1_var
- var_1

Invalid Identifiers

- !var1
- 1var
- 1_var
- var#1
- var 1

Python Comments

- **Comments in Python** are the lines in the code that are ignored by the interpreter during the execution of the program. Comments enhance the readability of the code and help the programmers to understand the code very carefully.
- There are three types of comments in Python:
  - Single line Comments
  - Multiline Comments
  - Docstring Comments

- Single line comment:

```
# sample comment
name =“RAM"
print(name)
```

- Multi line comments:

```
# Python program to demonstrate
# multiline comments
print("Multiline comments")
```

Python docstring is the string literals with triple quotes that are appeared right after the function. It is used to associate documentation that has been written with Python modules, functions, classes, and methods. It is added right below the functions, modules, or classes to describe what they do. In Python, the docstring is then made available via the __doc__ attribute.

```
def multiply(a, b):
    """Multiplies the value of a and b"""
    return a*b


# Print the docstring of multiply function
print(multiply.__doc__)
```

```
IN[1]:
   Welcome
   to Acads
IN[2]:
```

Python Shell Prompt

Outputs

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
```

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'
```

```
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'
```

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

+	-	*	//	/	%	**

==	!=	>	<	>=	<=

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

and	or	not

&	|	^	~	>>	<<

in	not in

is	is not

Variables

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

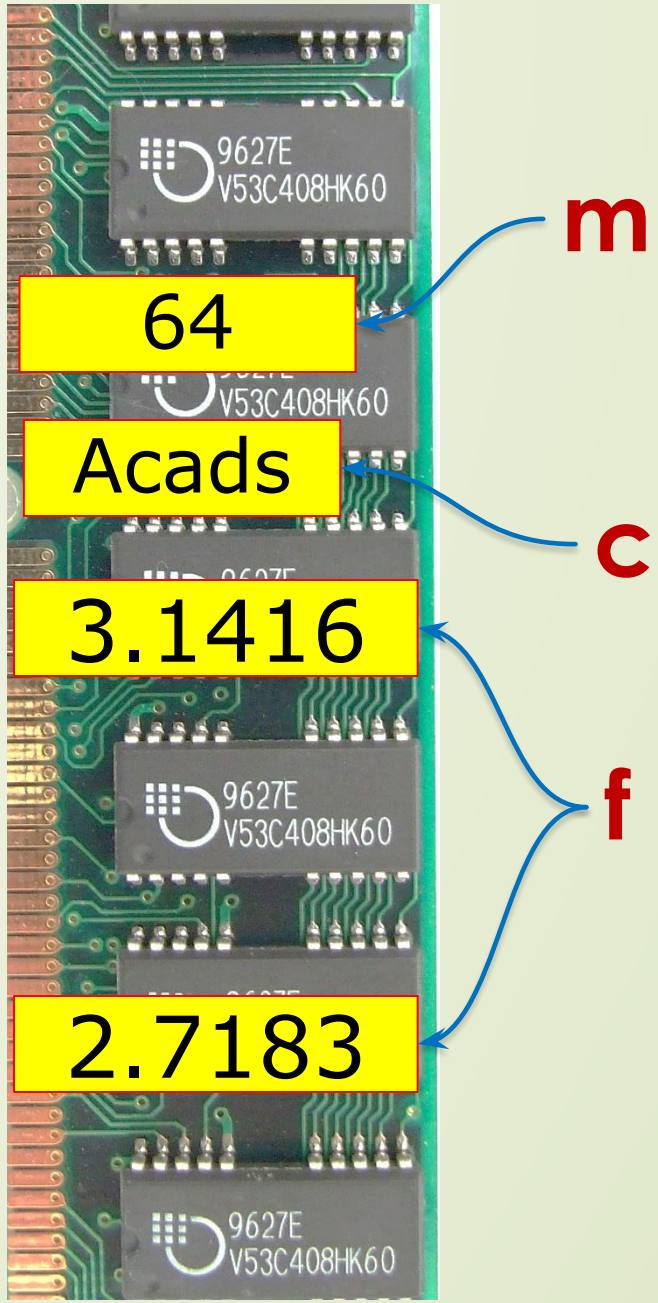
m = 64;

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

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

.6 is 0.4