Midterm Exam 1 on Thursday
Program Design

Input
Processing
Output
Determine requirements
↓
Write the source code
↓
Convert source code to object code
↓
Run the program
↓
Check the output
Python

High-level programming language

Developed in the 1990s by Guido van Rossum

Actively maintained and documented by programmers around the world

Clear syntax

General purpose usage

Wide range of libraries available

Python 3
Python Interpreter

Like all high-level programming languages, Python programs must be compiled and/or interpreted to run.

Python scripts have to be processed by another program called the “Python interpreter”.

The interpreter does the following:
- Reads your script
- Compiles it into “bytecode”
- Executes bytecode to run program
Basic Data Types

Integers

Floating Point Numbers

Strings
**Integer**

*int*

A whole number

Can be of unlimited size

Be careful not to use commas in your numbers, for example: 1,000
Floating point number

A number that contains a decimal point

Has minimum and maximum values

Limited precision

4 is not the same as 4.0
Arithmetic Operators
Lowest precedence to highest precedence

+    -
*    /
//
%
**
String

A sequence of one or more characters

A *string* of letters and numbers

Enclosed in quotation marks

Double quoted strings can contain single quotes and vice versa

Triple quotes can span multiple lines
String Operators

+ 
* 
#
**Variable**

A name that refers to a value

An “assignment statement” gives a value to a variable

Variables remember things

Variables can change, too

= is Python’s assignment token
Variable Names

Can be of any length
Characters must be letters, numbers, or the underscore ( `_ `)
First character cannot be a number
Case sensitive
Python keywords cannot be used as variable names
Multiple Assignment

Assigning more than one variable at a time is possible

```python
a, b, c = 'foo', 'bar', 'baz'
```

Swapping variable values

```python
x, y = y, x
```
Built-In Functions

A function is a reusable chunk of code

We’ve already been using functions

• print( )
• int( )
• type( )

Function name and arguments

Parentheses mean “execute this function”

Some functions take no input
Modules
A collection of related functions

To use a module, it must be imported:
import module

Then you can use any of the functions in that module:
module.function( )

You can also import all a module’s functions at once:
from module import *

This allows you to call the function without appending the module
Getting Help

Python has lots of documentation built in.

Functions and modules usually come with brief explanations.

To list functions in a module:
```
dir(module)
```

To see all of Python’s built-in functions:
```
dir(__builtins__)
```

To get help with a specific function:
```
help(function)
```

Python’s general help utility:
```
help()
```
Basic Input

Reading strings from the keyboard

Input function

```
variable = input('Prompt' )
```

Input can be in the form of a string or numeric data type
More on the Print Function

One of the most common functions we’ve been using

Important in our standalone Python programs for providing output

Separator
sep = '  '

End character
den = '\n'

Destination
file = open('file.txt', 'w')
Boolean Logic

Programming languages make decisions in binary terms

Two boolean values: True and False

True and False can be assigned to variables, just like strings and numbers

Logical operators for boolean values: ==, !=, and, or, and not
Equivalence

\[ a == b \]

Evaluates to True when \( a \) and \( b \) are the same:

- Both \( a \) and \( b \) are True
- Both \( a \) and \( b \) are False
Negation

\( a \neq b \)

Evaluates to True when \( a \) and \( b \) are *not* the same:

- \( a \) is True and \( b \) is False
- \( a \) is False and \( b \) is True
And

\( a \) and \( b \)

Evaluates to True when both \( a \) and \( b \) are True:

- \( a \) is True and \( b \) is True
Or

\(a \ or \ b\)

Evaluates to True when a is True or b is True:

- a is True and b is True
- a is True and b is False
- a is False and b is True
Boolean Logic and If-Statements

- **Not**

  - **not a**

  Evaluates to True when `a` is False and False when `a` is True:

  - `a` is False
  - `b` is False
# Truth Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a == b</th>
<th>a != b</th>
<th>a and b</th>
<th>a or b</th>
<th>not a</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Order of Execution
Highest to Lowest

a == b
a != b
not a
a and b
a or b
Brackets ( ) override operator precedence
Conditional Statements

Allow you to change the flow of a Python program

Consist of if-statements

Introduce code blocks
If/Else-Statements

if
else

Begins with the keyword if
Followed by a Boolean expression
May be followed by an else block for alternate conditions
If/Else If-Statements

if

elif

If-statement with more than one condition

Multiple elif blocks are permitted

else block is optional
Semantic Indentation

Code blocks are meaningful in Python

Indentation marks blocks of code and is not an optional format

Code blocks must be indented consistently by the same amount

As a result, Python code is clearer and more legible