CHAPTER 2
ELEMENTARY PROGRAMMING

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)
NEXT STEP IN PROGRAMMING

• Computations! Support for basic mathematics

• Imagine, computing interest on a loan, dividend on a stock, or even computing an angle to go to a specific location

• Here we make variables that store data and then alter those values which are stored.
EXAMPLE: COMPUTING THE AREA

```python
# Assign a value to radius
radius = 20

# Compute the area
area = radius * radius * 3.14159

# Display the result
print("The area for a circle with radius", radius, "is", area)
```

Let's practice tracing, but first a note on memory.
MEMORY

• Memory is storage for data and programs

• We will pretend that memory is an infinitely long piece of tape separated into different cells

• Each cell has an address, i.e., a location, and a value

• In the computer these values are represented in binary (0s and 1s) and addresses are located in hexadecimal (base 16, 0x)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0x1</td>
<td>0x2</td>
<td></td>
<td>0xA</td>
</tr>
</tbody>
</table>
EXAMPLE: COMPUTING THE AREA

ComputeArea.py
1. # Assign a value to radius
2. radius = 20
3.
4. # Compute the area
5. area = radius * radius * 3.14159
6.
7. # Display the result
8. print("The area for a circle with radius",
    radius, "is", area)

Output
The area for a circle with radius 20 is 1256.636

Memory
radius: 20
area: 1256.636

Actually operations are evaluated in a specific order. Temporary values are stored for these intermediate computations. print can output a series of values separated by a comma. Each value is separated by a space in the output.

A special symbol = gives a value to a variable, called assignment.
Reading input from the console

```
ComputeArea.py
1. # Assign a value to radius
2. radius = eval(input("Enter a value for a radius: "))
3. 
4. # Compute the area
5. area = radius * radius * 3.14159
6. 
7. # Display the result
8. print("The area for a circle with radius", radius, "is", area)
```

- `eval` is a function that converts those key strokes to a value.
- `input` is a function to collect key strokes from the console.
REPRESENTING DATA
WHAT ABOUT THE 0S AND 1S?

• Yes, computers operate in 0s and 1s. The python interpreter handles this business for us, but memory also stores values as 0s and 1s

• Memory also stores entirely 0s and 1s

• So what we need to know is how computers do this
INTEGER REPRESENTATION

• First, a look at our number system. It is base 10, meaning we use 10 different symbols (the digits). Let’s look at an example number: 1037.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^3</td>
<td>1000</td>
<td>10^2</td>
<td>100</td>
<td>10^1</td>
</tr>
<tr>
<td>1 * 10^3 +</td>
<td>0 * 10^2 +</td>
<td>3 * 10^1 +</td>
<td>7 * 10^0 = 1037</td>
<td></td>
</tr>
</tbody>
</table>

• And adding we use carry-and-add

```
   1 0 3 7
+ 0 0 4 9
```

```
1 0 8 6
```
INTEGER REPRESENTATION

• Synonymously, binary numbers work the same way. Except instead of base 10, it is base 2. A digit can only be 0 or 1. Example: 0010 0101

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^7 = 128$</td>
<td>$2^6 = 64$</td>
<td>$2^5 = 32$</td>
<td>$2^4 = 16$</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
<tr>
<td>0 * $2^7$ +</td>
<td>0 * $2^6$ +</td>
<td>1 * $2^5$ +</td>
<td>0 * $2^4$ +</td>
<td>0 * $2^3$ +</td>
<td>1 * $2^2$ +</td>
<td>0 * $2^1$ +</td>
<td>1 * $2^0 = 37$</td>
</tr>
</tbody>
</table>

• And adding 0010 1010 + 0000 0101

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• Note there are other common number systems: Octal (base 8, digits 0-7) and Hexadecimal (base 16, digits 0-9 and A-F, used for memory addresses)
INTEGER REPRESENTATION

• By limiting the number of bits, we limit the expressiveness of the data type
  • Means that if we only have 2 bits, we can only represent 4 numbers: 00, 01, 10, 11

• A 64 bit number can only represent values from \([-2^{64}, 2^{64})\]

• In programming, we must make conscious decisions about this otherwise there can be severe consequences
DATA TYPES

• Overall, data types define the available operations on and range of the data representation. Additionally, it notes how it is stored in memory.

• Right now we have seen:
  • Strings – sequences of characters, e.g., "Hello"
  • Floating point numbers – representing real numbers with fractional components, e.g., 3.54
  • Integers – representing positive and negative whole numbers, e.g., 15

• I want you to know about these, even though python will hide them from you and treat them fluidly
ACTIVITY

• With a partner

• Convert the binary number 1001 1001 to decimal

• Add the binary number 0101 0101 to 1001 1001 (DO NOT DO THIS IN DECIMAL) and then convert to a decimal number

• Bonus: 0xA1 to decimal, add 0x0E to it and convert to decimal. Hint: 0x means that the number is a hexadecimal (base 16)
VARIABLES AND NAMING
IDENTIFIERS (NAMES)

• An **identifier** is a sequence of characters that consist of letters, digits, underscores (_), and asterisk (*).

• An identifier must start with a letter or an underscore (_).

• An identifier cannot be a reserved word. (See Appendix A, “Python Keywords,” for a list of reserved words).

• An identifier can be of any length.
VARIABLES

• A **variable** is a named piece of data (memory). It stores a **value**.

• It has a **type** that defines how the memory is interpreted and what operations are allowed

\[ \text{var} = \text{value} \]
Expressions

- **Expressions** are combinations of literals, variables, operations, and function calls that generate new values

\[
\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)
\]

- is translated to

\[
(3+4\times x)/5 - 10\times (y-5)\times (a+b+c)/x + 9\times (4/x + (9+x)/y)
\]
ASSIGNMENT STATEMENTS

- Assignment statements give values to a variable
- \( x = 1; \) // Assign 1 to \( x \);
- \( \text{radius} = 1.0; \) // Assign 1.0 to \( \text{radius} \);
- \( \text{a} = 'A'; \) // Assign 'A' to \( \text{a} \);
SIMULTANEOUS ASSIGNMENT

• Python allows a shorthand to create/assign multiple variables at a time. Variables and expressions will be comma separated. An example:

\[ x, y = \frac{a+b}{2}, \frac{a-b}{2} \]
NAMED CONSTANTS

• Often, we need constants in programs, e.g., \(\pi\), whose value never changes.

• Python does not have a special syntax for naming constants. You can simply create a variable to denote a constant. To distinguish a constant from a variable, use all uppercase letters to name a constant.

• \(\text{PI} = 3.14159\)

• \(\text{SIZE} = 3\)
NAMING CONVENTIONS

• Choose meaningful and descriptive names.
• Typically begin with lower case
• Python typically names with underscores separating words (snake casing), but other styles capitalize the first letter of each subsequent word (camel casing):
  • my_area_variable
  • myAreaVariable
• Constants will be all caps using snake casing: MY_PI_CONSTANT
• Be consistent!
LITERALS

• A **literal** is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

• \( i = 34 \)
• \( x = 1000000 \)
• \( d = 5.0 \)
SCIENTIFIC NOTATION

• Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.
EXPRESSIONS
# NUMERIC OPERATORS

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
<td>34.0 − 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Float Division</td>
<td>1 / 2</td>
<td>0.5</td>
</tr>
<tr>
<td>//</td>
<td>Integer Division</td>
<td>1 // 2</td>
<td>0</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
<td>4 ** 0.5</td>
<td>2.0</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>
INTEGERS

DIVISION

• Integers do not store decimals
• Division computes how many times a divisor evenly goes into a number
• Remainder (modulus) computes what is left over
• $5 \div 2$ yields an integer 2
• $5 \% 2$ yields 1 (the remainder of the division)

Practice:
• What is $3456421 \% 2$?
• $25\%3$?
• $87\%4$?
HOW TO EVALUATE AN EXPRESSION

• Though Python has its own way to evaluate an expression behind the scene, the result of a Python expression and its corresponding arithmetic expression are the same.

• Therefore, you can safely apply the arithmetic rules for evaluating a Python expression.
UNDERFLOW AND OVERFLOW

- When a floating-point variable is assigned a value that is too large (in size) to be stored, it causes overflow, a run time exception. Example: $245^{1000}$

- When a floating-point number is too small (i.e., too close to zero) to be stored, it causes underflow. Python approximates it to zero. So normally you should not be concerned with underflow.

Questions
- Why does overflow/underflow occur?
- What types of applications would we care about them?
EXERCISE

• Write a program to compute the average of three numbers
• Trace the execution in memory if the user enters 3, 5, 7
# Augmented Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>Addition assignment</td>
<td>i += 8</td>
<td>i = i + 8</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
<td>i -= 8</td>
<td>i = i - 8</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
<td>i *= 8</td>
<td>i = i * 8</td>
</tr>
<tr>
<td>/=</td>
<td>Float division assignment</td>
<td>i /= 8</td>
<td>i = i / 8</td>
</tr>
<tr>
<td>//=</td>
<td>Integer division assignment</td>
<td>i //= 8</td>
<td>i = i // 8</td>
</tr>
<tr>
<td>%=</td>
<td>Remainder assignment</td>
<td>i %= 8</td>
<td>i = i % 8</td>
</tr>
<tr>
<td>**=</td>
<td>Exponent assignment</td>
<td>i **= 8</td>
<td>i = i ** 8</td>
</tr>
</tbody>
</table>
TYPE CONVERSION
TYPE CONVERSION

• Use `int()`, `float()`, `str()` to convert any type to integer, floating-point, or string respectively.

• Consider the following statements and their results:
  • `int(4.7) → 4`
  • `float(4) → 4.0`
  • `str(4) → "4"

• To round floating point numbers use `round()`
  • `round(4.7) → 5`
EXERCISE

• Write a program to compute sales tax for a purchase.
• How could you alter your program to only store 2 decimal places? Try it!
• Trace your program with a purchase of $100
EXERCISE

• With turtle graphics:
  • Write a program that asks the user to enter an \((x, y)\) coordinate representing the center of a rectangle. Additionally, ask the width and height of the rectangle.
  • Draw the rectangle.
CHAPTER 3
MATH, STRINGS, AND OBJECTS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)
MATH MODULE

• Python provides many useful mathematics methods in its Math module for performing common mathematical functions.

• In order to use them we need to understand:
  • What a function is
  • How to use functions
  • Where we look up possible functions to use
EXAMPLES OF MATH MODULE

\[ \text{max}(2, 3, 4) \] # Returns a maximum number : in this case 4
\[ \text{min}(2, 3, 4) \] # Returns a minimum number : in this case 2
\[ \text{round}(3.51) \] # Rounds to its nearest integer
\[ \text{round}(3.4) \] # Rounds to its nearest integer
\[ \text{abs}(-3) \] # Returns the absolute value
\[ \text{pow}(2, 3) \] # Same as 2 ** 3
# The Math Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>fabs(x)</td>
<td>Returns the absolute value of the argument.</td>
<td>fabs(-2) is 2</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>Rounds x up to its nearest integer and returns this integer.</td>
<td>ceil(2.1) is 3</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Rounds x down to its nearest integer and returns this integer.</td>
<td>floor(2.1) is 2</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Returns the exponential function of x (e^x).</td>
<td>exp(1) is 2.71828</td>
</tr>
<tr>
<td>log(x)</td>
<td>Returns the natural logarithm of x.</td>
<td>log(2.71828) is 1.0</td>
</tr>
<tr>
<td>log(x, base)</td>
<td>Returns the logarithm of x for the specified base.</td>
<td>log10(10, 10) is 1</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Returns the square root of x.</td>
<td>sqrt(4.0) is 2</td>
</tr>
<tr>
<td>sin(x)</td>
<td>Returns the sine of x. x represents an angle in radians.</td>
<td>sin(3.14159 / 2) is 1</td>
</tr>
<tr>
<td>asin(x)</td>
<td>Returns the angle in radians for the inverse of sine.</td>
<td>asin(1.0) is 1.57</td>
</tr>
<tr>
<td>cos(x)</td>
<td>Returns the cosine of x. x represents an angle in radians.</td>
<td>cos(3.14159 / 2) is 0</td>
</tr>
<tr>
<td>acos(x)</td>
<td>Returns the angle in radians for the inverse of cosine.</td>
<td>acos(1.0) is 0</td>
</tr>
<tr>
<td>tan(x)</td>
<td>Returns the tangent of x. x represents an angle in radians.</td>
<td>tan(0.0) is 0</td>
</tr>
<tr>
<td>fmod(x, y)</td>
<td>Returns the remainder of x/y as double.</td>
<td>fmod(2.4, 1.3) is 1.1</td>
</tr>
<tr>
<td>degrees(x)</td>
<td>Converts angle x from radians to degrees.</td>
<td>degrees(1.57) is 90</td>
</tr>
<tr>
<td>radians(x)</td>
<td>Converts angle x from degrees to radians.</td>
<td>radians(90) is 1.57</td>
</tr>
</tbody>
</table>
STRINGS AND CHARACTERS

• A string is a sequence of characters. String literals can be enclosed in matching single quotes ('') or double quotes (""). Python does not have a data type for characters. A single-character string represents a character.

• Strings have many methods to use to manipulate their data

Letter = 'A' # Same as letter = "A"
numChar = '4' # Same as numChar = "4"
message = "Good morning" # Same as message = 'Good morning'
THE STRING CONCATENATION OPERATOR

• You can use the + operator add two numbers. The + operator can also be used to concatenate (combine) two strings. Here are some examples:

• message = "Welcome " + "to " + "Python"
To read a string from the console, use the `input` function. For example, the following code reads three strings from the keyboard:

```python
s = input("Enter a string: ")
print("s is " + s)
```
EXAMPLES OF STRING OBJECT METHODS

• s = "Welcome"
• s1 = s.lower() # Invoke the lower method, stores 'welcome'
• s2 = s.upper() # Invoke the upper method, stores 'WELCOME'
Another useful string method is `strip()`, which can be used to strip the whitespace characters from the both ends of a string.

- `s = "\t Welcome \n"
- `s1 = s.strip()` # Invoke the `strip` method, `s1` stores 'Welcome'
• **Methods** are subroutines that we would like to (re)use again and again in code

• For example, would you like a method to compute $\sqrt{x}$ or write a lengthy algorithm every time you wish to use it?

• Python provides many useful methods. Some we have seen:
  • `print()`, `input()`, `round()`
INTERPRETING FUNCTIONS/METHODS

• Consider the following from the Math library:

\[ \text{sqrt}(x) \]

• \text{sqrt} is an identifier, i.e., a name, for this method

• \(x\) is called a parameter, or an argument. This is the input to the method.

• Methods can optionally output data too, in this case it will output a number.

• In a few weeks, we will learn to write our own methods. For now, we need to know how to use them.
INTRODUCTION TO OBJECTS AND METHODS

• In Python, all data—including numbers and strings—are actually objects.

• An object is an entity. Each object has an id and a type. Objects of the same kind have the same type. You can use the id function and type function to get these information for an object.
INVOKING A METHOD

• There is a difference between these math functions and how we used `turtle`.

• Methods sometimes depend on the value of an object/class and sometimes they do not. Common math functions, like `sqrt`, do not need to know anything besides the parameter. However, other things like `turtle` needs to know where the turtle currently is, so we invoke methods from a variable instead:

• `turtle.forward(100)`  //Use the variable
FORMATTING NUMBERS AND STRINGS

• Often it is desirable to display numbers in certain format. For example, the following code computes the interest, given the amount and the annual interest rate.

• The format function formats a number or a string and returns a string.

   \texttt{format(item, format-specifier)}
FORMATTING FLOATING-POINT NUMBERS

\[
\text{format}(57.467657, '10.2f') \\
\text{format}(12345678.923, '10.2f') \\
\text{format}(57.4, '10.2f') \\
\text{format}(57, '10.2f')
\]

Field width: 10
Precision: 2
Conversion code: f
Format specifier: 10.2f

10
57.47
12345678.92
57.40
57.00
• If you change the conversion code from f to e, the number will be formatted in scientific notation. For example

```python
format(57.467657, '10.2e')
format(0.0033923, '10.2e')
format(57.4, '10.2e')
format(57, '10.2e')
```
**FORMATTING AS A PERCENTAGE**

- You can use the conversion code % to format numbers as a percentage. For example,

```python
format(0.53457, '10.2%')
format(0.0033923, '10.2%')
format(7.4, '10.2%')
format(57, '10.2%')
```
JUSTIFYING FORMAT

• By default, the format is right justified. You can put the symbol < in the format specifier to specify that the item is left justified in the resulting format within the specified width. For example,

\[
\text{format}(57.467657, '10.2f')
\]
\[
\text{format}(57.467657, '<10.2f')
\]
You can use the conversion code `s` to format a string with a specified width. For example,

```python
format("Welcome to Python", '20s')
format("Welcome to Python", '<20s')
format("Welcome to Python", '>20s')
```

```
Welcome to Python
Welcome to Python
□□□Welcome to Python
```

```python
Welcome to Python
Welcome to Python
□□□Welcome to Python
```
EXERCISE

• Recall there is more information online and in your book

• I assume you know what is in Ch. 3 of your book

• Write a program that prompts the user to enter a side length and an angle from \((0^\circ, 90^\circ)\) and draws a right triangle accordingly. Label each side length and each angle using the turtle module.