ELEMENTARY PROGRAMMING

INTERNATIONAL COLLEGE, KMITL

PRESSESIONAL COURSE

OVERALL

- Writing a simple program
- Reading input from console
- Identifiers
- Variables, Assignments, Expression
- Named constant
- Numeric Data Types and Operators
- Operators precedence
- Augmented Assignment Operators
- Type Conversion and Rounding

WRITING A SIMPLE PROGRAM

GETTING STARTED

- An **algorithm** describes how a problem is solved.
- An algorithm can be described in natural languages or **pseudocode** (a natural language mixed with some programming code).

TASK

- Write a program that calculated the area of the circle from the user entered radius.

ALGORITHM

1. Get the circle's radius from the user.
2. Compute the area by applying the following formula:

$$Area = \pi \times radius \ \times radius$$

3. Display the result.

PROBLEM

- Retrieving input from the user
- Storing the radius in the program

READING USER INPUT

FROM CONSOLE

READING USER INPUT :: RESERVED WORD

LISTING 2.2 ComputeAreaWithConsoleInput.py

input radius

compute area

display result

```
# Prompt the user to enter a radius
radius = eval(input("Enter a value for radius: "))

# Compute area
area = radius * radius * 3.14159

# Display results
print("The area for the circle of radius", radius, "is", area)
```

reserved word

```
Enter a value for radius: 2.5 ↵Enter
The area for the circle of radius 2.5 is 19.6349375
```

```
Enter a value for radius: 23 ↵Enter
The area for the circle of radius 23 is 1661.90111
```

** Both input() and eval() are reserved word

READING USER INPUT :: INPUT()

- You can use input() to receive input from console:

variable = input(“some string display“)

- The default datatype of the variable is string
- The parameter is display in the program.

```
>>> variable = input("some string display:  ")
some string display:  I love programming
>>> variable
'I love programming'
>>>
```

READING USER INPUT :: EVAL()

- When valued entered is a string.
- eval() evaluates and converts string to a numerical values.

```
s = input("Enter a value for radius: ")  # Read input as a string
radius = eval(s) # Convert the string to a number
```

- For example:
  - eval(“34.5”) return 34.5

IDENTIFIERS

IDENTIFIERS

- Names of things appear in the program.
- A sequence of characters consists only of letters, digits and underscore(_)
- Must start with letter or underscore (_)
- Cannot be a keyword (reserved word)
- Can be of any length

IDENTIFIERS

```python
# Prompt the user to enter a radius
radius = eval(input("Enter a value for radius: "))

# Compute area
area = radius * radius * 3.14159

# Display results
print("The area for the circle of radius", radius, "is", area)
```

```
Enter a value for radius: 2.5 ↵Enter
The area for the circle of radius 2.5 is 19.6349375
```

```
Enter a value for radius: 23 ↵Enter
The area for the circle of radius 23 is 1661.90111
```

** python is case-sensitive **

IDENTIFIERS :: EXAMPLE

- print
- 2num
- This+is+the+end

Invalid identifiers

- circle_radius
- circleRadius

Good programming style identifiers

VARIABLES, ASSIGNMENTS, EXPRESSION

ASSIGNMENT

Assignment operator (=)

```
# Assign a value to radius
radius = 20 # radius is now 20

# Compute area
area = radius * radius * 3.14159

# Display results
print("The area for the circle of radius", radius, "is", area)
```

radius → 20

area → 1256.636

```
The area for the circle of radius 20 is 1256.636
```

The statement for assigning a value to a variable is called an **assignment statement.** The equal sign (=) is used as the **assignment operator.**

ASSIGNMENT

```
y = 1                        # Assign 1 to variable y
radius = 1.0                 # Assign 1.0 to variable radius
x = 5 * (3 / 2) + 3 * 2      # Assign the value of the expression to x
x = y + 1                    # Assign the addition of y and 1 to x
area = radius * radius * 3.14159 # Compute area
```

```
x = x + 1
```

```
1 = x     # Wrong
```

The syntax for assignment statement :

```
variable = expression
```

ASSIGNMENT

```
i = j = k = 1
```

```
k = 1
j = k
i = j
```

- Both are equivalent.
- Precedence of assignment operator is from right to left

ASSIGNMENT :: CAUTION

```
>>> count = count + 1
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    count = count + 1
NameError: name 'count' is not defined
```

Error
count is not initialized

```
>>> count = 1
>>> count
1
>>> count = count + 1
>>> count
2
```

Fixed
count can now be incremented

SIMULTANEOUS ASSIGNMENT (SWAP)

```
>>> x = 1
>>> y = 2
>>> temp = x # Save x in a temp variable
>>> x = y # Assign the value in y to x
>>> y = temp # Assign the value in temp to y
>>> x
2
>>> y
1
```

```
>>> x = 1
>>> y = 2
>>> x, y = y, x # Swap x with y
>>> x
2
>>> y
1
```

** Difference methods yield the same result **

CONSTANT

CONSTANT :: ADVANTAGE

- Don’t have to repeatedly type the same value if it is used multiple time.
- Need to change the value inly in a single location
- Descriptive names make the program easy to read

CONSTANT :: EXAMPLE

```
>>> PI = 3.141592654
>>> r1 = 2
>>> r2 = 3
>>> area1 = PI * (r1 ** 2)
>>> area2 = PI * (r2 ** 2)
>>> area1
12.566370616
>>> area2
28.274333886
```

NUMERIC DATA TYPES AND OPERATORS

NUMERIC OPERATOR

TABLE 2.1 Numeric Operators

Name	*Meaning*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

- The **operands** are the values operated by the operator.
- The **unary operator** required one operand.
- The **binary operator** required twos

NUMERIC OPERATOR

```
>>> 4 / 2
2.0
>>> 2 / 4
0.5
>>>
```

Float Division : /

```
>>> 5 // 2
2
>>> 2 // 4
0
>>>
```

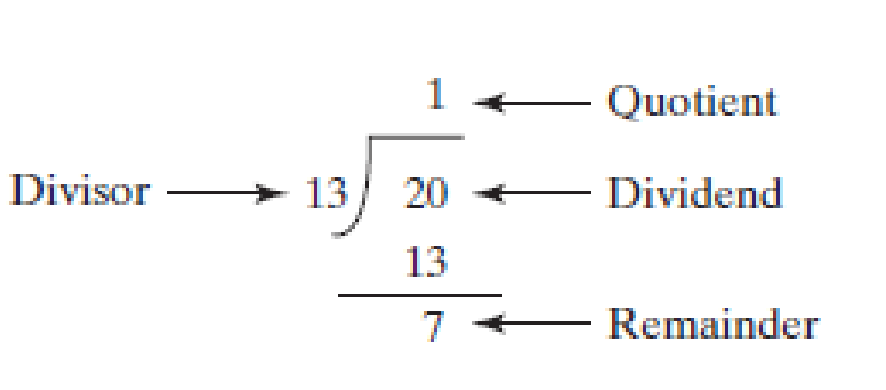
Integer Division : //

```
>>> 2.3 ** 3.5
18.45216910555504
>>> (-2.5) ** 2
6.25
>>>
```

Exponentiation : **

MODULO OPERATOR

- The % operator known as, remainder or modulo operator, yields the remainder of the division




```
>>> 20/13
1.5384615384615385
>>> 20//13
1
>>> 20%13
7
```

OPERATOR PRECEDENCE

OPERATOR PRECEDENCE

**

* / // %

\+ -

OPERATOR PRECEDENCE

```
3 + 4 * 4 + 5 * (4 + 3) - 1
                        ↑ (1) inside parentheses first
3 + 4 * 4 + 5 * 7 - 1
        ↑ (2) multiplication
3 + 16 + 5 * 7 - 1
             ↑ (3) multiplication
3 + 16 + 35 - 1
  ↑ (4) addition
19 + 35 - 1
   ↑ (5) addition
     54 - 1
        ↑ (6) subtraction
     53
```

AUGMENTED ASSIGNMENT OPERATORS

AUGMENTED ASSIGNMENT OPERATORS

TABLE 2.2 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

TYPE CONVERSION AND ROUNDING

TYPE CONVERSION AND ROUNDING

```
>>> value = 5.6
>>> int(value)
5
>>>
```

```
>>> value = 5.6
>>> round(value)
6
>>>
```

```
>>> value = 5.6
>>> round(value)
6
>>> value
5.6
>>>
```