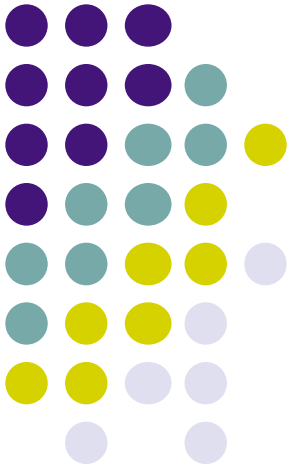
ISYS90088 Introduction to Application Development



Week 3 –

Continued from week 2- arithmetic precedence, examples for mixed type arithmetic, fundamentals of strings – concatenation and length, Boolean and logical operations

If time permits – introduction to if statement

Semester 2 , 2018

Dr Antonette Mendoza

Objectives

- Continued from week 2 - arithmetic precedence, examples for mixed type arithmetic
- fundamentals of strings – concatenation, and length,
- Boolean and logical operations
- If time permits – introduction to if statement

Recap-Arithmetic Expressions

- An **arithmetic expression** consists of operands and operators combined in a manner that is already familiar to you from learning algebra

OPERATOR	MEANING	SYNTAX
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

Arithmetic Expressions (continued)

- **Precedence rules**:
  - ** has the highest precedence and is evaluated first
  - Unary negation is evaluated next
  - *, /, and % are evaluated before + and -
  - + and - are evaluated before equal to (=)
  - With two exceptions, operations of equal precedence are **left associative**, so they are evaluated from left to right
    - Exponentiation (**) and assignment (=) are **right associative**
  - You can use parenthesis ( ) to change the order of evaluation as parenthesis takes precedence.

Precedence rule for arithmetic operators (continued)

TYPE OF OPERATOR	OPERATOR SYMBOL
Exponentiation	**
Arithmetic negation	-
Multiplication, division, remainder	*, /, %
Addition, subtraction	+, -

Arithmetic Expressions (continued)

EXPRESSION	EVALUATION	VALUE
5 + 3 * 2	5 + 6	11
(5 + 3) * 2	8 * 2	16
6 % 2	0	0
2 * 3 ** 2	2 * 9	18
-3 ** 2	-(3 ** 2)	-9
-(3) ** 2	9	9
2 ** 3 ** 2	2 ** 9	512
(2 ** 3) ** 2	8 ** 2	64
45 / 0	Error: cannot divide by 0	
45 % 0	Error: cannot divide by 0	

This should be -9

Syntax error: set of rules for constructing well formed expressions in a language (error when an expression or sentence is not well formed).

Semantic error: detected when the action that an expression describes cannot be carried out, even if the expression is syntactically correct.

Example: 45%0 is a **semantic error**

Arithmetic calculations – Quiz

#Let x = 8 and y = 2. Write the values of the following expressions:

a. x + y * 3
b. (x + y) * 3
c. x ** y
d. x % y
e. x / 12.0
f. x // 6
g. 3 + 4 ** 2//5

Mixed-Mode Arithmetic and Type Conversions

- **Mixed-mode arithmetic** involves integers and floating-point numbers:

```
>>> 3.14 * 3 ** 2
28.26
```


- **Remember**—Python has different operators for quotient and exact division:

`3 // 2 * 5.0` yields `1 * 5.0`, which yields `5.0`

`3 / 2 * 5` yields `1.5 * 5`, which yields `7.5`

Tip:

  - Use exact division
  - Use a **type conversion function** with variables

Example: type conversion

Check example on IDLE =>Using print; input and type conversion

Using some useful library functions in python

- Python includes many useful functions, which are organized in libraries of code called **modules**
  - **What is a function?** A **function** is chunk of code that can be called by name to perform a task
  - Functions often require **arguments** or **parameters**
    - Arguments may be **optional** or **required**
  - When function completes its task, it may **return a value** back to the part of the program that called it.

Note: Functions will be taught in detail later in the course

help, math modules in python – to help start simple programming

```
>>> help(round)

Help on built-in function round in module builtin:

round(...)
    round(number[, ndigits]) -> floating point number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the same type as
    number. ndigits may be negative.
```



```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atanh', 'ceil','copysign', 'cos', 'cosh', 'degrees', 'e',
'exp', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'hypot',
'isinf', 'isnan', 'ldexp', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow',
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Example - math Module

- To use a resource from a module, you write the name of a module as a qualifier, followed by a dot (.) and the name of the resource
  - Example usage with a qualifier – **math.pi**


```
>>> math.pi
3.1415926535897931
>>> math.sqrt(2)
1.4142135623730951
```

```
>>> from math import pi, sqrt
>>> print(pi, sqrt(2))
3.14159265359 1.41421356237
>>>
```

- You may import all of a module's resources to use without the qualifier
  - Example: **from math import ***

Mixed-Mode Arithmetic and Type Conversions - Examples

```
>>> int(6.75)

>>>round(6.51)

>>> round(6.75)

>>> float(4)

>>>int(“33”)

>>>str('a' + chr(ord('d')))
```

Mixed-Mode Arithmetic and Type Conversions (continued)

- Type conversion also occurs in the construction of strings from numbers and other strings


```
>>> profit = 1000.55
>>> print('$' + profit)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'float' objects
```


- Solution: use **`str`** function

```
>>> print('$' + str(profit))
$1000.55
```

- Python is a strongly **typed** programming language

String Literals

- A string literal is a sequence of characters enclosed in single or double quotation marks
- '' and "" represent the **empty string**
- Use ''' and """ for multi-line paragraphs


```
>>> "I'm using a single quote in this string!"
"I'm using a single quote in this string!"
>>> print("I'm using a single quote in this string!")
I'm using a single quote in this string!
>>>
>>> print("""This very long sentence extends all the way to
the next line.""")
This very long sentence extends all the way to
the next line.
```

```
>>> """This very long sentence extends all the way to
the next line. """
'This very long sentence extends all the way to\nthe next line.'
>>>
```

String Concatenation

- You can join two or more strings to form a new string using the concatenation operator **+**
- The * operator allows you to build a string by repeating another string a given number of times


```
>>> " " * 10 + "Python"
'          Python'
>>>
```

Quiz

write the output of the following python statements:

a. “hell_no”
b. “hell_no” * 10
c. “hell_no” + “ “ * 10
d. (“hell_no” + “ “) * 10

String length

Using a library function to count the length of a string:

- String's length - Number of characters it contains (0+).
- **len** is a library function that allows us to do some manipulation with strings.


```
>>> len("Hi there!")
9
>>> len("")
0
```

Example – to show!

Escape Characters

- It is a special character that is preceded with a backslash(\) appearing inside a string literal.
- When a string literal that contains the escape character is printed, the escape characters are treated as special commands that are embedded in the string.

\n	new line character
\t	horizontal tab
\\	character \
\’	single quotation mark
\”	double quotation mark

Escape Characters – some useful ones

Examples:

```
>>> print ('mon\ttues\twed')
>>>print('mon\ntues\nwed')
>>>print('a\\b')
>>>print('a\"b')
>>>print('\'b')
```

The Boolean Type, Comparisons, and Boolean Expressions

- **Boolean data type** consists of two values: true and false (typically through standard **True/False**)

COMPARISON OPERATOR	MEANING
==	Equals
!=	Not equals
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

[TABLE 3.2] The comparison operators

- Example: 4 != 4 evaluates to False

Logical Operators and Compound Boolean Expressions (continued)

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

A	not A
True	False
False	True

[FIGURE 3.4] The truth tables for **and**, **or**, and **not**

Logical Operators and Compound Boolean Expressions (continued)

- Next example verifies some of the claims made in the previous truth tables:


```
>>> A = True
>>> B = False
>>> A and B
False
>>> A or B
True
>>> not A
False
```


- The logical operators are evaluated after comparisons but before the assignment operator
  - **`not`** has higher precedence than **`and`** and **`or`**

Logical Operators and Compound Boolean Expressions (continued)

TYPE OF OPERATOR	OPERATOR SYMBOL
Exponentiation	`**`
Arithmetic negation	`-`
Multiplication, division, remainder	`*`, `/`, `%`
Addition, subtraction	`+`, `-`
Comparison	`==`, `!=`, `<`, `>`, `<=`, `>=`
Logical negation	`not`
Logical conjunction and disjunction	`and`, `or`
Assignment	`=`

[TABLE 3-4] Operator precedence, from highest to lowest

Logical operation evaluation: Example

- In **(A and B)**, if **A** is false, then so is the expression, and there is no need to evaluate **B**
- In **(A or B)**, if **A** is true, then so is the expression, and there is no need to evaluate **B**

Quiz

Fill in the blanks:

- A compound boolean expression created with a the ---------- operator is true only if both of its sub expressions are true:

Is it the or, and, not???

The -------- operator takes a boolean expression as its operand and reverses its logical value.

Is it the or, not or and operators?????

- Break (if time permits)

Selection: if and if-else Statements

- **Selection statements** allow a computer to make choices - based on a **condition**

- The **if** statement is used to create a decision structure, which allows a program to have more than one path of execution.

- The **if** statement causes one or more statements to execute only when a Boolean expression is true.

- It is **a control structure** – a logical design that controls the order in which a set of statements execute.

The Boolean Type, Comparisons, and Boolean Expressions

- Boolean data type consists of two values: true and false (typically through standard `True`/`False`)

COMPARISON OPERATOR	MEANING
==	Equals
!=	Not equals
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

- Example: `4 != 4` evaluates to `False`

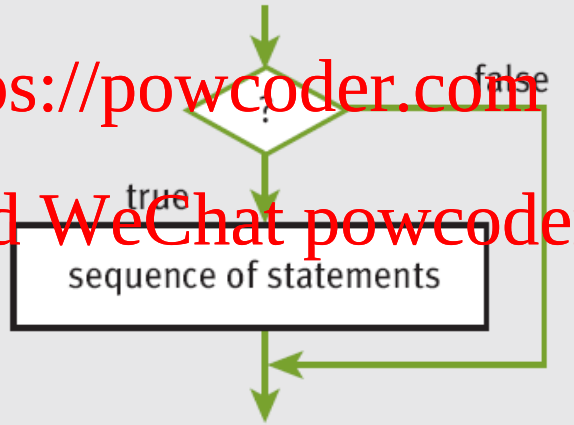
The **if – else** statement

- The simplest is a **one-way selection** statement (if)
- Also called a **two-way selection** statement (if-else)
- The condition in the if-else statement must be a Boolean expression – that is, an expression that evaluates to either true or false

One-Way Selection Statements

- Simplest form of selection is the ***if* statement**

```
if <condition>:
    <sequence of statements>
```


```
>>>x = -2
>>>if x < 0:
        x = abs(x)
>>>print(x)
```

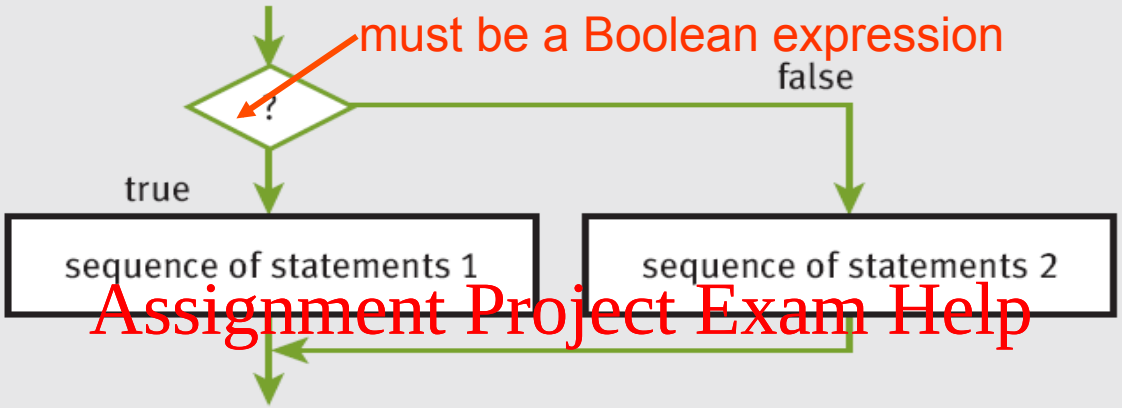
if-else Statements

- The two possible actions each consist of a sequence of statements
- Each sequence must be indented at least one space beyond the symbols if and else.

Syntax:

```
if <condition>:
    <sequence of statements-1>
else:
    <sequence of statements-2>
```

if-else Statements (continued)




```
first = int(input("Enter the first number: "))
second = int(input("Enter the second number: "))
if first > second:
    maximum = first
    minimum = second
else:
    maximum = second
    minimum = first
print("Maximum:", maximum)
print("Minimum:", minimum)
```

Exercise - if-else Statements (continued)

Write a program to accepts two positive integers and print out the maximum and the minimum of the two input numbers - using max and min functions from **the math library**