PYTHON BASICS AND FLOW CONTROL

CS 3080 Python Programming

UCCS University of Colorado Colorado Springs

Credits: Damia Fuentes Escote and Dana Wortman

Today's class

- Python basics – Chapter 1
- Flow control – Chapter 2

UCCS

PYTHON BASICS

Expressions

- **Expression**
  - *Combinations of **values** and **operators** that can **evaluate** down to a single value*
  - *2 + 2*
  - *4 * 3*
  - *5 – 4 * (3 + 1)*
  - *2*

- 2 + 2 is evaluated down to a single value, 4.
- A single value with no operators is also considered an expression, though it evaluates only to itself.

Operators

Operator	Operation	Example	Evaluates to…
**	Exponent	`2 ** 3`	8
%	Modulus/remainder	`22 % 8`	6
//	Integer division/floored quotient	`22 // 8`	2
/	Division	`22 / 8`	2.75
*	Multiplication	`3 * 5`	15
-	Subtraction	`5 - 2`	3
+	Addition	`2 + 2`	4

Precedence

- The ** operator is evaluated first;
- the * , / , // , and % operators are evaluated next, from left to right;
- the + and - operators are evaluated last (also from left to right).
- You can use parentheses ( ) to override the usual precedence if you need to.
- Python will keep evaluating parts of the expression until it becomes a single value

```
(5 - 1) * ((7 + 1) / (3 - 1))
    ↓
4 * ((7 + 1) / (3 - 1))
    ↓
4 * (  8    / (3 - 1))
    ↓
4 * (  8    /    2   )
    ↓
4 * 4.0
    ↓
16.0
```

Operator	Operation	Example	Evaluates to...
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6
//	Integer division/floored quotient	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 2	4

Common Data Types

- A data type is a category for values, and every value belongs to exactly one data type
- The meaning of an operator may change based on the data types of the values next to it.
  - *String replication operator*
    - 'Alice' * 5

Table 1-2: Common Data Types

Data type	Examples
Integers	`-2, -1, 0, 1, 2, 3, 4, 5`
Floating-point numbers	`-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25`
Strings	`'a', 'aa', 'aaa', 'Hello!', '11 cats'`

Storing values in variables

- You'll store values in variables with an **assignment statement**. An assignment statement consists of a variable name, an equal sign (called the **assignment operator**), and the value to be stored.
- A variable is **initialized** (or created) the first time a value is stored in it
- **Overwriting** the variable -> When a variable is assigned a new value, the old value is forgotten. It doesn't matter the data type.
- There is no variable declaration in Python! You just assign a value to a variable and it comes into existence. But never use a variable name without having assigned a value
- **Multiple assignment**
  - *a = b = c = 1*
  - *a,b,c = 1,2,"john"*

I do NOT recommend using multiple assignment (can be confusing and lead to mistakes), unless you know what you doing

Variable names

Underscores are a special-use character in classes, so don't use them to start variable names!

- 3 rules:
  - *It can be only one word, i.e., no spaces.*
  - *It can use only letters, numbers, and the underscore (_) character.*
  - *It can't begin with a number.*

Valid variable names	Invalid variable names
`balance`	`current-balance` (hyphens are not allowed)
`currentBalance`	`current balance` (spaces are not allowed)
`current_balance`	`4account` (can't begin with a number)
`_spam`	`42` (can't begin with a number)
`SPAM`	`total_$um` (special characters like `$` are not allowed)
`account4`	`'hello'` (special characters like `'` are not allowed)

Variable names

- Variable names are **case-sensitive**
  - *spam , SPAM , Spam , and sPaM are four different variables.*
- It is a Python convention to start your variables with a lowercase letter.
- Descriptive names makes your code more readable.
- **Camelcase**
  - *lookLikeThis*
- **Underscores** (official Python code style)
  - *looking_like_this*
- It doesn’t matter which way, but be consistent in your code!

Comments

- Python ignores comments, and you can use them to write notes or remind yourself what the code is trying to do.
- Any text for the rest of the line following a hash mark (#) is part of a comment.
  - *# This is a one line comment*
- Any text in between three delimiters (''') is part of a comment. Used to explain things in more detail.
  - '''

    This is a multiline comment.

    Second line.

    '''

Basic functions

- The **print()** function displays the string value inside the parentheses on the screen.
- The **input()** function waits for the user to type some text on the keyboard and press enter. *Always returns a string*.
- The **len()** function evaluates to the integer value of the number of characters in the string argument (or a variable containing a string).
  - *len('hello world') evaluates to 11*
- The **str()** function can be passed an integer value and will evaluate to a string value version of it.
  - *str(29) evaluates to '29'*
- The **int()** function can be passed a string value and will evaluate to an int value version of it.
  - *int('29') evaluates to 29*
  - *The int() function is also useful if you need to round a floating-point number down*

First program

- Ask the user for his name and age.
- Then, tell him:
  - *The length of his name*
  - *How old will be in a year (= age + 1)*

```
print('Hello world!')


# ask for their name
print('What is your name?')
myName = input()
print('It is good to meet you, ' + myName)

# print the length of their name
print('The length of your name is:')
print(len(myName))

# ask for their age
print('What is your age?')
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Why do we need to convert an int to a string (try: remove it and see the error)?

FLOW CONTROL

Flow control

- A program is just a series of statements.
- **Flow control statements** can decide which Python statements to execute under which conditions.
- Based on how the expressions evaluate, the program can decide to skip instructions, repeat them, or choose one of several instructions to run.
  - *if, else, elif*
  - *while loops*
  - *for loops*
- All **flow control statements** end with a colon and are followed by a new block of code (the clause)

UCCS

Boolean data type

- Only two values:
  - *True*
  - *False*
- spam = True
- is_over_21 = False

Comparison Operators

- **Comparison operators** compare two values and evaluate down to a single Boolean value

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

= is different from == :
- = is the assignment statement
- == equal to operator

UCCS

Boolean operators

- and, or, and not are used to compare Boolean values.
- Like comparison operators, they evaluate these expressions down to a Boolean value.

Expression	Evaluates to...
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`

Expression	Evaluates to...
`True or True`	`True`
`True or False`	`True`
`False or True`	`True`
`False or False`	`False`

Expression	Evaluates to...
`not True`	`False`
`not False`	`True`

UCCS

Conditions

- All the expressions with Boolean operators can be considered conditions.
- **Conditions** always evaluate down to a Boolean value, True or False.
  - *(4 < 5) and (5 < 6)* *# True*
  - *(1 == 2) or (2 == 2)* *# True*

Blocks of code

- Lines of Python code can be grouped together in **blocks**. You can tell when a block begins and ends from the indentation of the lines of code.

```
   if name == 'Mary':
❶      print('Hello Mary')
       if password == 'swordfish':
❷          print('Access granted.')
       else:
❸          print('Wrong password.')
```

In most editors…

Highlight code blocks and press TAB to indent

Press SHIFT-TAB to un-indent

UCCS

if statements

- An **if** statement's clause (that is, the block following the if statement) will execute if the statement's condition is True.
- The clause is skipped if the condition is False.
- An **if** statement consists of the following
  - The **if** keyword
  - A condition (an expression that evaluates to True or False)
  - A colon
  - Starting on the next line, an indented block of code (called the **if** clause)

else statements

- An **if** clause can *optionally* be followed by an **else** statement
- The **else** clause is executed only when the if statement's condition is False
  - An **else** statement doesn't have a condition
- An **else** statement consists of the following
  - The **else** keyword
  - A colon
  - Starting on the next line, an indented block of code (called the **else** clause)

elif statements

- The **elif** statement is an “else if” statement that always follows an **if** or another **elif** statement.
  - Used when you want one of *many* possible clauses to execute
- Provides another condition that is checked only if all of the previous conditions were False
- An **elif** statement consists of the following
  - The **elif** keyword
  - A condition (expression that evaluates to True or False)
  - A colon
  - Starting on the next line, an indented block of code (called the **elif** clause)
- Not guaranteed that at least one of the clauses will be executed
  - When there is a chain of **elif** statements, only one or none of the clauses will be executed
  - Once one of the statements’ conditions is found to be True, the rest of the **elif** clauses are automatically skipped

while loop statements

- The code in a **while** clause will be executed over and over again as long as the while statement's condition is True
- A **while** statement consists of the following

  The **while** keyword

  A condition (expression that evaluates to True or False)

  A colon

  Starting on the next line, an indented block of code (called the **while** clause)

break statements

- If the execution in a while reaches a **break** statement, it immediately exits the while loop's clause.

continue statements

- When the program execution reaches a **continue** statement, the program execution immediately jumps back to the start of the loop and reevaluates the loop's condition. (This is also what happens when the execution reaches the end of the loop.)

“Truthy” and “Falsey” values

- Conditions will consider some values in other data types as equivalent to True and False
  - When used in conditions, **0**, **0.0**, and **''** (the empty string) are considered False, while all other values are considered True.

for loop statement

- The for **loop** is used to execute a block of code only a certain number of times.
- You can use break and continue statements too.
- A **for** statement looks something like *for I in range(5)*: and includes the following
  - *The **for** keyword*
  - *A variable name*
  - *The **in** keyword*
  - *A call the the **range()** method with up to three integers passed to it*
  - *A colon*
  - *Starting on the next line, an indented block of code (called the **for** clause)*

Importing modules

- Python comes with a set of modules called the **standard library**. Each module is a Python program that contains a related group of functions that can be embedded in your programs.
- For example, the math module has mathematics related functions, the random module has random number–related functions, and so on.
- Once you import a module, you can use all the cool functions of that module.

```
import random
print(random.randint(1, 10))
```

Importing modules

- Multiple imports

  import *random, sys, os, math*

- from import statements. No need to add the random. prefix. Two ways:
  - ***from*** *random* ***import*** *randint*

    *print(****randint(1, 10)****)*

  - *from random import **

    *print(****randint(1, 10)****)*

Terminate the program early

```
import sys
while True:
        print('Type exit to exit.')
        response = input()
        if response == 'exit':
                sys.exit()
        print('You typed ' + response + '.')
```