Introduction to Coding

Tan Sein Jone

University of British Columbia

August 7, 2024

Table of contents

1. Functions
2. Variables
3. Data Types
4. Conditionals
5. Loops

Table of Contents

Hello World

- The first program that every programmer writes
- How to start a Python program
- How to print to the console

What is a Function?

- A function is a block of code that performs a specific task
- A function can take in parameters
- A function can return a value

Why Use Functions?

- Functions make code more modular
- Functions make code more readable
- Functions make code more reusable

Calling a Function

- A function can ba called within a program/script
- A function can also be called using the terminal

In The Terminal

```
python3 print("Hello,␣World!")
```

Breakdown

- python3: The Python interpreter
- print(): The function
- "Hello, World!": The argument
- Output: This command will print "Hello, World!" to the console

Why Running Code Like This is Not Ideal

- Code is not saved (in other words, it’s ephemeral)
- Code is not reusable
- Code is not readable (everything is in one line)
- Let’s instead try to run the code in a script

In The Terminal

```
touch first_notebook.ipynb
```

Breakdown

- touch: Command to create a file
- first_notebook.ipynb: The name of the file
- Output: This command will create a file called first_notebook.ipynb

Jupyter Notebooks vs Python Scripts

- They both run Python code
- Jupyter Notebooks are more interactive
- Jupyter Notebooks are more visual
- Jupyter Notebooks are more user-friendly, due to the ability to run code in cells
- It makes it easier to debug code

Running Code in a Jupyter Notebook

- Open the Jupyter Notebook
- Create a new cell
- Write the code in the cell
- Run the cell

first_notebook.ipynb

```
print(”Hello,␣World!”)
```

Functions and Packages/Libraries

- Functions can be defined in packages/libraries
- Functions can be imported from packages/libraries
- Functions can also be built into the Python interpreter
- Certain functions and classes are reserved words

Common Built-in Functions

- print(): Prints to the console
- input(): Takes user input
- len(): Returns the length of an object
- range(): Returns a sequence of numbers
- type(): Returns the type of an object

Expressive Languages

- Python is what's known as an expressive language
- Expressive languages are designed to be easy to read and write
- These languages translate code into machine code
- An example of machine code is binary

Table of Contents

1. Functions
2. Variables
3. Data Types
4. Conditionals
5. Loops

What are Variables?

- Variables are used to store data
- Variables are assigned a value
- Variables can be changed

Variable Naming Rules

- Variables must start with a letter or underscore
- Variables can only contain letters, numbers, and underscores
- Variables are case-sensitive
- Variables cannot be reserved words

Variable Naming Conventions

- Camel Case: myVariableName
- Pascal Case: MyVariableName
- Snake Case: my_variable_name

Variable Naming Conventions

- Each language has its own naming conventions
- Python uses snake case
- JavaScript uses camel case
- C# uses pascal case

Variable Naming Tips

- Try to name variables descriptively
- But don't make it so descriptive that it's long and hard to read
- Bear in mind YOU will have to be the one typing these variables out
- Make sure the variable name is relevant to the data it's storing

Good and Bad Examples of Variable Names

- Good: name, age, grade
- Too short and ambiguous: n, a, g
- Too long and descriptive: name_of_student, age_of_student, grade_of_student
- Too long and irrelevant: name_of_student_in_class, age_of_student_in_class, grade_of_student_in_class

Variables and Modern IDEs

- Modern IDEs have features that help with variable naming and autocompletion
- Most IDEs have a feature similar to intellesense in Visual Studio Code
- This feature will suggest variable names as you type
- It will also bring up a list of variables that have also been created

Pay Attention to the Warnings Your IDE Gives You

- IDEs will give you warnings if you use a variable that hasn't been declared
- IDEs will give you warnings if you use a variable that has already been declared
- IDEs will give you warnings if you use a variable that is not being used

Scope

- Global Variables: Variables declared outside of a function
  - Can be accessed anywhere
- Local Variables: Variables declared inside of a function
  - Can only be accessed within the function

```
def my_function()\:
    x = 10
x = 20
my_function()
print(x)
```

Breakdown

- x = 20: This is a global variable
- x = 10: This is a local variable
- print(x): This will print 20
- This is because the x in the function is a local variable
- The x outside of the function is a global variable

Reusing Variable Names

- In general, it's best to avoid reusing variable names, especially if they're part of different scopes
- This can lead to confusion
- This can lead to errors
- You can however get away with this if you import functions from different scripts

Table of Contents

1. Functions
2. Variables
3. Data Types
4. Conditionals
5. Loops

Data Types

- Integers: Whole numbers
- Floats: Numbers with decimals
- Strings: Text
- Booleans: True or False
- Lists: Ordered collection of items
- Tuples: Ordered collection of items that cannot be changed
- Dictionaries: Unordered collection of items
- Sets: Unordered collection of unique items

Type Checking

- Type checking is used to determine the data type of a variable
- Type checking is used to ensure that the correct data type is being used
- Type checking is used to prevent errors

Importance of Type Checking

- There are advantages and disadvantages to using each data type

- Interacting with different data types can cause errors

Common Data Type Errors

- Mixing data types
- Using the wrong data type
- Not converting data types

Why Not Store An Int as a String?

- It's not efficient
- An int takes up less memory than a string
- You cannot perform mathematical operations on a string

```
random_int = 10
random_float = 10.0
random_string = ”10”
```

Type Casting

- Converting between data types
- Can be done using built-in functions
- Not all conversions are possible

Common Type Casting Errors

- Converting a string to an int that is not a number
- Converting a float to an int that is not a whole number
- Converting a float to an int that is too large

Common Data Type Conversions

- int(): Converts a value to an integer
- float(): Converts a value to a float
- str(): Converts a value to a string
- bool(): Converts a value to a boolean

```
converted_int = int(random_float)
```

Data Types with Multiple Values

- Lists, Tuples, Dictionaries, and Sets can store multiple values
- Each value can be a different data type
- Each value can be accessed using an index
- Each value can be changed

Common Data Types with Multiple Values

- Lists: Ordered collection of items
- Tuples: Ordered collection of items that cannot be changed
- Dictionaries: Unordered collection of items
- Sets: Unordered collection of unique items

```
random_list = [10, 10.0, ”10”]
random_tuple = (10, 10.0, ”10”)
random_dict = {”int”: 10, ”float”: 10.0, ”string”: ”10”}
random_set = {10, 10.0, ”10”}
```

Common Usecases

- Lists and dicitonaries will likely be the most used data types
- Lists are used to store multiple values
- Dictionaries are used to store key-value pairs

When to Use a List

- When you need to store multiple values
- You need to change the values
- You have to keep track of the order of the values

When to Use a Dictionary

- When you need to store key-value pairs
- You need to change the values
- You don’t need to keep track of the order of the values

Common Methods for Lists

- append(): Adds an element to the end of the list
- insert(): Adds an element at a specific index
- remove(): Removes an element from the list
- pop(): Removes an element at a specific index
- clear(): Removes all elements from the list

Common Methods for Dictionaries

- get(): Gets the value of a key
- keys(): Gets all the keys
- values(): Gets all the values
- items(): Gets all the key-value pairs
- clear(): Removes all key-value pairs

Why Don’t We ALways Use Dictionaries?

- Dictionaries are not ordered
- Dictionaries are not indexed
- Dictionaries are not iterable

Combining Data Types

- Data types can be combined
- Lists can store dictionaries
- Dictionaries can store lists

Lists of dictionaries

- Say for example we have a list of students
- Each student has a name, age, and grade
- We can store this information in a list of dictionaries
- Each dictionary will represent a student

```
students = [
    {”name”: ”Alice”, ”age”: 20, ”grade”: 90},
    {”name”: ”Bob”, ”age”: 21, ”grade”: 85},
    {”name”: ”Charlie”, ”age”: 22, ”grade”: 80}
]
```

Dictionaries of lists

- Say for example we have a dictionary of students
- Each student has a list of grades
- We can store this information in a dictionary of lists
- Each key will represent a student

```
students = {
    ”Alice”: [90, 85, 80],
    ”Bob”: [85, 80, 75],
    ”Charlie”: [80, 75, 70]
}
```

Possible Problem

- Assuming that the order of the list of grades is important
- We cannot guarantee that the order of the list of grades will be maintained
- This is because dictionaries are not ordered

Possible Solution

- We can use a dictionary of dictionaries instead
- Each key will represent a student
- Each value will be a dictionary of grades

Table of Contents

What are Conditionals?

- Conditionals are used to make decisions
- Conditionals are used to execute code based on a condition
- Conditionals are used to compare values

Comparison Operators

- ==: Equal to
- !=: Not equal to
- <: Less than
- >: Greater than
- <=: Less than or equal to
- >=: Greater than or equal to

Common Use Cases

- If a student's grade is greater than 90, print "A"
- If condition is true, execute code
- If condition is false, execute other code

Logical Operators

- and: Returns True if both statements are true
- or: Returns True if one of the statements is true
- not: Returns True if the statement is false

If Statements

- If statements are used to execute code if a condition is true
- If statements can be followed by an else statement
- If statements can be followed by an elif statement

```
x = 10
if x == 10:
    print(”x␣is␣10”)
elif x == 20:
    print(”x␣is␣20”)
else:
    print(”x␣is␣not␣10␣or␣20”)
```

Breakdown

- x = 10: This is the value of x
- if x == 10: This is the condition
- print(”x is 10”): This is the code that will be executed if the condition is true
- elif x == 20: This is the condition that will be checked if the first condition is false
- print(”x is 20”): This is the code that will be executed if the condition is true

Common Mistakes

- Not using the correct comparison operator
- Not using the correct logical operator
- Not using the correct indentation

Nested If Statements

- If statements can be nested
- Nested if statements are used to check multiple conditions
- Nested if statements can be difficult to read

Alternatives to Nested If Statements

- Using elif statements
- Using logical operators
- Using functions

Table of Contents

What are Loops?

- Loops are used to repeat code
- Loops are used to iterate over a sequence
- Loops are used to execute code a specific number of times

Common Types of Loops

- For Loops: Used to iterate over a sequence
- While Loops: Used to execute code as long as a condition is true

For Loops

- For loops are used to iterate over a sequence
- For loops are used to execute code a specific number of times
- For loops can be used with lists, tuples, dictionaries, and sets

```
for x in range(10):
    print(x)
```

Breakdown

- for x in range(10): This is the loop
- print(x): This is the code that will be executed
- x: This is the variable that will be used to iterate over the sequence
- range(10): This is the sequence

While Loops

- While loops are used to execute code as long as a condition is true
- While loops are used to execute code a specific number of times
- While loops can be used with lists, tuples, dictionaries, and sets

```
x = 0
while x < 10:
    print(x)
    x += 1
```

Breakdown

- x = 0: This is the variable that will be used to check the condition
- while x ¡ 10: This is the loop
- print(x): This is the code that will be executed
- x += 1: This is the code that will increment the variable

Common Mistakes

- Not using the correct comparison operator
- Not using the correct logical operator
- Not using the correct indentation

Nested Loops

- Loops can be nested
- Nested loops are used to iterate over multiple sequences
- Nested loops can be difficult to read