Week 2

0.1 Outline

This week, we take a deeper dive into basic data types in Python, conditional statements and loops, and simple function definitions.

- Expressions
- Strings
- Assignments and Bindings
- Function definitions (basics)
- `if-elif-else` statements
- `for` loops
- `while` loops

0.2 Bookmark these sites!

The official Python Documentation. From here you can navigate to:

- The Python Tutorial
- The Python Standard Library
- The Python Module Index

0.3 Expressions

Expressions are composed from variables, values and **operators**. Some examples of operators are:

- Arithmetic operators: `+`, `-`, `/` (true division), `//` (integer division leaving a quotient), `*`, `%` (remainder in integer division) etc.
- Logical operators: `and`, `or`, `not` etc.
- Sequence operators: `[]` (indexing or slicing), `+` (concatenation), `*` (replication) etc.
- Call expressions: using previously defined **functions**.

> **Note**
>
> Datatypes are implemented in Python as **classes**. Functions that are associated with a datatype value are called **methods**.

0.3.1 Exercise: Leap Year Calculation

A year is a **leap** year if it is divisible by 4 but not divisible by 100 unless it is divisible by 400. For instance, 2004 is a leap year, 1900 is not, but 2000 is.

Suppose that a variable `year` holds an integer that represents a year. Construct an expression that would be true if and only if `year` is a leap year.

```
# Ask user to input a year and store the integer in variable `year`

# Construct the expression for `year` being a leap year
```

0.4 Expression Evaluation

The Python interpreter **evaluates** expressions in the context of an **environment**.

> **Environment:** Collection of **bindings** of names to values.

Python's environment is **dynamic**: every added definition or assignment changes it!

All **builtin** bindings (e.g., functions like `len`, `bin`, `str`, `int`) are available to the interpreter in the **main environment**.

0.5 Strings

Python's strings are **Unicode** strings: symbols from all known languages, glyphs and synthesized symbols can be used in a Python string! For instance: `'\u03C0'`, `'\U0001f644'`

Internally, strings are **encoded** into bytes, e.g., using the UTF-8 encoding. Printing a string involves **decoding** the string.

> **Note**
>
> Data can occur in all kinds of formats/encodings. It is important to understand some of the history of encodings: even if you might not understand some of the details, please check out Ned Batchelder's PyCon 2012 presentation on Unicode in Python.

- **Immutable**: Operations on string operands create *new* strings.

Examples of string operations: length, concatenation, replication, indexing, slicing.

0.5.1 Exercise

```
# Start with the string x whose value is 'Data '.

# Find its length.

# Obtain the string y with value 'Data Science' from x

# Replicate y 5 times

# Find the index of the second occurrence of 'a' in y
```

```
# Find the string made up of every other character from y
```

A **program** in Python is a sequence of statements that are evaluated from top to bottom.

> **!** Important
>
> Evaluation of a statement causes one or more of the following effects:
> - change in state of existing variable(s): **new binding(s)**
> - addition of new variables to the current environment's namespace: **new definitions**
> - creation of a **new frame** as a result of a **function call**

0.6 Assignment Statement

> Tip
>
> Use the Python Tutor website to visualize the **binding** of values to names in a Python code snippet!

- Assignments bind names to values
- Binding is **dynamic**: the type of a value always remains unchanged, but the same name may refer to different values (and hence different types) at different times during the program execution.

0.6.1 Exercise

Compute the approximate acceleration of gravity for an object above the earth's surface, assigning `accel_gravity` with the result. The expression for the acceleration of gravity is: $(G \cdot M)/(d^2)$, where G is the gravitational constant 6.673×10^{-11}, M is the mass of the earth 5.98×10^{24} (in kg), and d is the distance in meters from the Earth's center (stored in variable `dist_center`).

Your code should ask for the distance to be entered as input, and should print the acceleration accurate to 2 decimal places.

0.7 `import` statement

Main purpose: augment the *current namespace* with additional name bindings obtained from the module being imported.

- add definitions of constants, functions and data types (classes) from built-in Python libraries
- add auxiliary code developed for an application

> Warning
>
> Avoid **namespace pollution**: use qualified names via **dot** notation, e.g., `math.pi`

0.7.1 Exercise

Import the `fractions` module.

- Use the `dir` builtin function to see what is defined in the module
- Use the `help` builtin function to get **documentation** on a specific definition, e.g. the `fractions.Fraction` datatype
- Experiment with different operations on fractions

0.8 Function Definition

Encapsulates a piece of computation whose outcome (**return value**) depends on the supplied inputs (**parameters** or **arguments**)

Separation of code into functional units allows us to:

- **compose** functions in a *modular* fashion
- create clean interfaces between parts of code
- practice the **DRY** principle (Don't repeat yourself)
- **test** code in small, manageable units

0.8.1 Exercise

- Define a function that computes the hypotenuse of a right-angle triangle given its legs by using the Pythagorean theorem.
- Follow documentation guidelines: create a **docstring**
- Use the `doctest` module to create simple tests for the function within the docstring.

1. The `pass` statement is a **placeholder** and does nothing!
2. Never test two real (i.e. `float`) values for equality! Use `math.isclose` instead.

```
def hypotenuse(leg_1, leg_2):
    pass
```

0.9 Selection statement: `if-elif-else`

Sometimes referred to as a **multi-way branch**.

```
if <condition 1>:
    <code block 1>
elif <condition 2>:
    <statement block 2>
...
else:
```

```
    <default code block>
```

- `<code block 1>` is **only evaluated** if `<condition 1>` is True
- In general, `<code block n>` is **only evaluated** if **all** the conditions for preceding blocks are `False` and `<condition n>` is `True`
- If control reaches `else`, `<default code block>` is executed.
- Control flows out of the selection statement after a selected code block (or the default block) is executed.

0.9.1 Exercise

The surface gravity on the Moon is 1.62 m/s^2; on Mars, it is 3.711 m/s^2; and on Earth it is 9.807 m/s^2. Write a Python script that asks the user to enter either "Moon", "Earth" or "Mars" at a prompt, and computes the weight in Newtons of a 180 lb. person on that planetary body.

0.10 `for` Loops

```
for <var> in <iterable collection>:
    <code block>
```

- `<var>` is also called the **loop control target**.
- At the start of each iteration, `<var>` is bound to the **next value** from the `<iterable collection>`. When there is no such value available, control flows out of the loop.
- During the iteration, the `<code block>` of code is executed. Then control flows back to the beginning of the loop.

0.10.1 Exercise

Write a function that takes a non-negative integer as an argument. It should return a count of the number of even digits in the number.

```
def count_even_digits(num):
    """Returns number of even digits in num

    Args:
        num (int): a non-negative integer

    Returns:
        int
    """
    pass
```

0.10.2 Exercise: Reading test files

Write a program that reads a text file `alice_wonderland.txt` in the same directory as the program (e.g., the notebook). It should print the percentage of lines that mention **Alice**.

> **Note**
>
> The builtin `open` function can be used to open a text file for reading from or writing into the file. The resulting object provides an **iterable collection** of **lines** in the file.

0.11 `while` loop

This implememts **indefinite** iteration controlled by a boolean condition (called a **predicate**):

```
while <condition>:
    <code block>
```

- `<condition>` is tested at the beginning of an iteration
- if `True`, the iteration proceeds by executing `<code block>`. Flow of control then reverts back to beginning of the loop.
- if `False`, control flows out of the loop

0.12 `break` and `continue` statements

`break` or `continue` statements may occur inside a loop body.

- `break`: Immediately **terminates** the nearest enclosing loop: if it is a `for` loop, the loop control target keeps its current value.
- `continue`: Forces control to revert to the start of the next iteration of the loop.

0.12.1 Exercise

Define a function that takes as input parameters a text file and a string. The function should print the first line, if any, that contains the given string as a substring. It should return the number of lines read before such a line (if it exists) is found.

> **Warning**
>
> Filenames, when written as strings, are often **paths** in the underlying operating system. It is bad practice to pass such paths to functions that expect name of files! Do so only when you know that the file is in the same directory as the program.

```
def find_string(filename, phrase):
    """Returns the number of lines in the file prior to the first occurrence of phrase.

    The function also prints the first line that contains the phrase

    Args:
        filename (str): name of text file in the program's directory
        phrase (str): the phrase being searched

    Returns:
        int
    """
    pass
```

0.12.2 Exercise

Define a function that takes as argument a non-negative integer and returns the sum of the **individual digits** of its argument.

```
def sum_of_digits(n):
    """Returns the sum of the digits on n

    Args:
        n (int): non-negative integer

    Returns:
        int
    """
    pass
```

0.12.3 Exercise

From Wikipedia:

> The Collatz conjecture is a conjecture in mathematics that concerns a sequence defined as follows: start with any positive integer n. Then each term is obtained from the previous term as follows: if the previous term is even, the next term is one half of the previous term. If the previous term is odd, the next term is 3 times the previous term plus 1. The conjecture is that no matter what value of n, the sequence will always reach 1.

Ask the user to input a non-negative integer x, and determine the length of x's Collatz sequence (which starts at x and ends at 1).

0.12.4 Exercise

Use the `random` module to compute the approximate value of π.

Intution: Throw darts randomly (but uniformly) in a square with side length 2. Count the proportion of darts that land in the **incircle** for the square (i.e., the inscribed circle).