Programming for Essential Digital Skills, Part 1

Christoph Walsh

2024

What is Python?

- Python is a *programming language* which specializes in statistical computing and graphics.
- A programming language is a way to instruct a computer to perform operations via written text.
- When programming, we need to be very exact. Otherwise the computer will throw and error or do something we didn't intend it to do!
- Spyder is a desktop application where you can write Python code, execute Python programs, and view plots created by Python.

Why Learn Python?

Python has many benefits over some alternatives:

- It is free and open source.
- Large active community creating packages and providing support.
- Easier to learn than some alternatives.
- Extremely versatile. For example, these slides were made with Python!
- Most common language used in data science.

Installing Python

- To download and install Python and Spyder, go to https://www.anaconda.com/download.
- Follow all the default settings.
- Afterwards, open Spyder.

The IPython Console

- We can perform calculations in the IPython Console tab in Spyder.
- At the most basic level, we can use it as a simple calculator:
  - Add: `2 + 3`
  - Subtract: `5 - 3`
  - Multiply: `2 * 3`
  - Divide: `3 / 2`
  - Combining operations: `(2 + 4) / (4 * 2)`
  - Exponentiation: `2 ** 3`

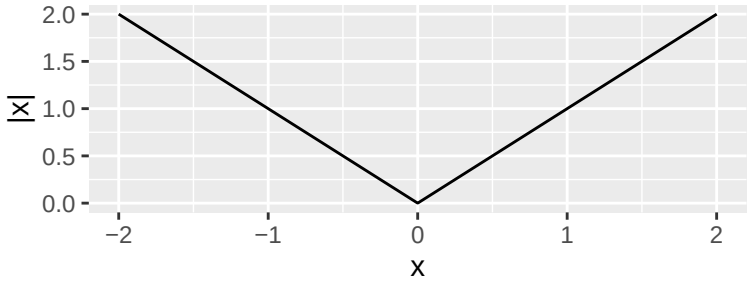
Python Functions

- Just like in Excel, Python has functions.
- These functions work in a very similar way:
  - Functions have names, and we provide *arguments* to functions inside parentheses.
- In the next few slides we will see some examples of mathematical functions and how to evaluate them in Python.

Absolute Value

- The absolute value function turns negative numbers into positive ones and has no effect on zero or positive numbers.

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases}$$



Absolute Value in Python

- The absolute value function in Python is called `abs`. We can use it as follows:

```
abs(-2)
```

2

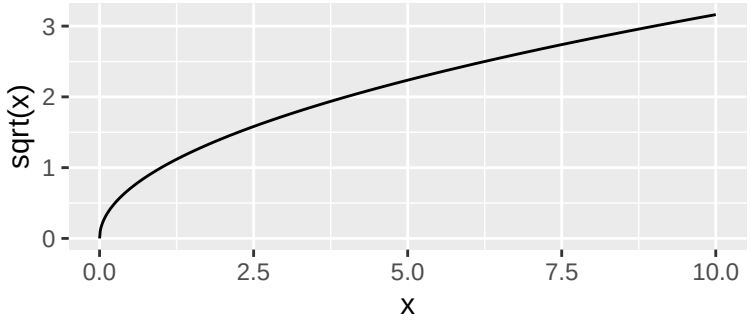
```
abs(3)
```

3

- We can see help about a function using `help(abs)`

Principal Square Roots

- The square root of a number is the y that solves $y^2 = x$.
- If $x = 4$, both $y = -2$ and $y = 2$ solve $y^2 = x$.
- The principal square root is the positive y solving this.



Principal Square Roots in Python

- Python doesn't have a square root function built in. But we take a number to the power of $\frac{1}{2}$ to take the square root:

```
9 ** (1/2)
```

```
3.0
```

- The cubed root is the y that solves $y = x^3$. We can also calculate this in Python by taking the power of $\frac{1}{3}$.

```
8 ** (1/3)
```

```
[1] 2
```

The math Module in Python

- Although there is no square root function built in, there is one in the `math` module.
- We can load this module and all its functions using `import math`. We can then using its square root function with:

```
import math
math.sqrt(9)
```

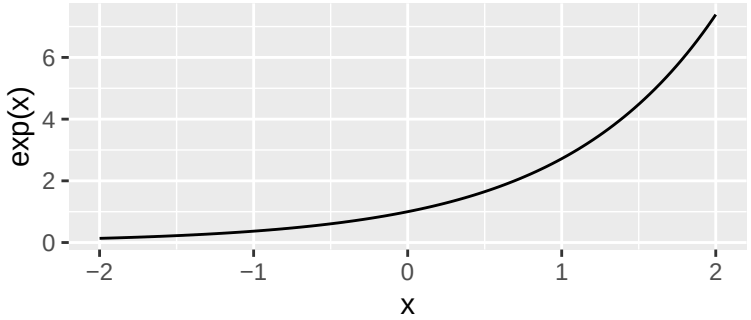
```
3.0
```

The Exponential Function

The exponential function is a very common function in statistics:

$$e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

Note: you don't need to know this function for the exam.



The Exponential Function in Python

- In Python we use the `math.exp()` function to calculate the exponential of a number:

```
import math
math.exp(0)
```

```
1.0
```

```
math.exp(1)
```

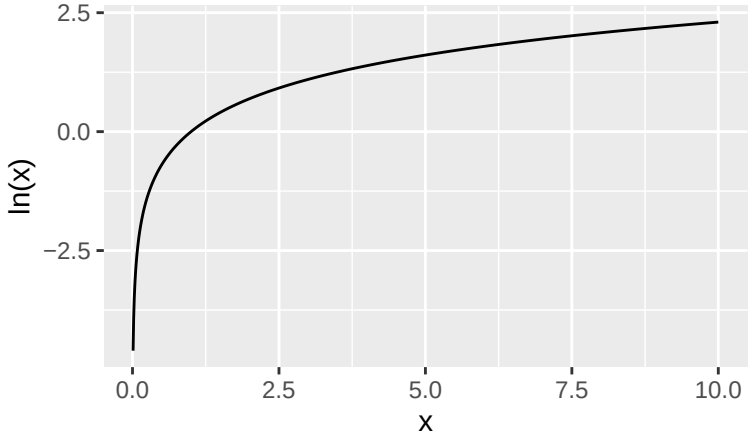
```
2.718281828459045
```

The Logarithm

- Another common mathematical function is the logarithm, which is like the reverse of exponentiation.
- The log of a number x to a base b, denoted $\log_b(x)$, is the number of times we need to multiply b by itself to get x.
- For example, $\log_{10}(100) = 2$, because $10 \times 10 = 100$. We need to multiply the base $b = 10$ by itself twice to get to $x = 100$.

The Natural Logarithm

- A special logarithm is the natural logarithm, $\log_e(x)$, which is the logarithm to the base $\exp(1) = e^1 \approx 2.7183$. This is also written as $\ln(x)$.



The Logarithm in Python

- In Python we use the `math.log()` function to calculate the natural logarithm:

```
math.log(1)
```

```
0.0
```

- We can calculate the logarithm of a number to a different base using the `base` argument. For example, for $\log_{10}(100)$:

```
math.log(100, 10)
```

```
2.0
```

Integer Division and the Modulus Operator

- Dividing $\frac{7}{3}$ can also be expressed as “2 with remainder 1”. We can get this in Python with *integer division* (with `//`) and the *modulus operator* (with `%`).
- The integer division operator always rounds down to the nearest whole number:

```
7 // 3
```

```
2
```

- The modulus operator gives the remainder when dividing two numbers:

```
7 % 3
```

```
1
```

- We’ll see that the modulus operator is extremely useful in programming.

Assigning Values to Variables

- We can assign *values* to *variables* using *assignment*.
- For example:

```
a = 2
b = 3
```

- a and b are then visible in the **Variable Explorer** tab in Spyder.
- We can then use a and b for calculations:

```
a + b
```

```
5
```

- There are rules for naming variables:
  - Must start with either a letter or _ and contain no spaces.
  - Variable names cannot be a keyword in Python.

Data Types for Single Values

- Integers (`int`): whole numbers, positive or negative.
- Floating-Point Numbers (`float`): real-valued numbers.
  - Be careful with floating-point precision!
- Strings (`str`): sequences of characters, e.g. `'abc'`.
  - Addition combines strings: `'abc' + 'def'`
  - Multiplication by an integer repeats strings: `3 * 'abc'`
- Boolean (`bool`): The Logical constants `True` and `False`.

Logical Operators

There are 3 logical operators: `and`, `or` and `not`:

- `a and b` is `True` if both `a` and `b` are `True`. Otherwise it is `False`.
- `a or b` is `True` if either `a` or `b` are `True`. Otherwise it is `False`.
- `not a` is `True` if `a` is `False`. Otherwise it is `True`.

```
a = True
b = False
a and b
```

```
False
```

```
a or b
```

```
True
```

```
not a
```

```
False
```

Comparison Operators

We can also check weak/strict (in)equalities in Python:

```
a = 2
b = 3
a == b  # is a equal to b?
```

```
False
```

```
a != b  # is a not equal to b?
```

```
True
```

```
a > b   # is strictly greater than b?
```

```
False
```

```
a <= b  # is weakly less than b?
```

```
True
```

Type Conversion

We can sometimes convert between `int`, `float`, `str` and `bool`, but we don't always get intuitive results:

```
float('1.0')
```

```
1.0
```

```
str(1)
```

```
'1'
```

```
int(True)
```

```
1
```

```
bool('False')
```

```
True
```

Assigning Multiple Values to a Variable

- Python has a number of ways you can assign multiple values to a single variable:
  - Lists
  - Tuplies
  - Dictionaries
  - Sets (and Frozen Sets).

Lists

Lists can contain any sequence of values, which could be integers, floats, strings, Booleans, or any combination of them. They can even contain lists as elements.

```
a = [2, 4, 6]
a = [1, 1.1, True, "hello"]
a = [1, 2, [3, 4]]
```

Arithmetic Operations on Lists

`+` combines lists and `*` repeats lists (like with strings):

```
a = [2, 4, 6]
b = [8, 10, 12]
a + b
```

```
[2, 4, 6, 8, 10, 12]
```

```
3 * a
```

```
[2, 4, 6, 2, 4, 6, 2, 4, 6]
```

`-` and `/` do not work on lists.

Indexing

- We can extract elements of a list with indexing, which uses the order the elements appear.
- In Python (and many other languages like C), indexing starts at 0.
  - This means the 1st element has index 0, the 2nd element has index 1, and so on.

```
a = [2, 4, 6]
a[0]
```

2

```
a[1]
```

4

```
a[2]
```

6

Mutability of Lists

We can change the value of an element using index:

```
a = [2, 4, 6]
a[0] = 8
a
```

```
[8, 4, 6]
```

Because lists have this property we say they are *mutable*.

List Slicing

To get a range of elements in a list we can using *slicing*:

```
a = [1, 2, 3, 4, 5]
a[1:3]
```

```
[2, 3]
```

```
a[:2]
```

```
[1, 2]
```

```
a[2:]
```

```
[3, 4, 5]
```

```
a[:]
```

```
[1, 2, 3, 4, 5]
```

List Methods

Lists have *methods*, which are like functions but instead of providing the list as an argument to the function, we apply the method directly to the object.

We can add an element at the end of a list using the `append()` method:

```
a = [2, 4, 6]
a.append(8)
a
```

```
[2, 4, 6, 8]
```

List Methods

Removing an element (by index):

```
a = [2, 4, 6]
a.pop(1)
```

```
4
```

```
a
```

```
[2, 6]
```

Sorting a list:

```
a = [6, 2, 4]
a.sort()
a
```

```
[2, 4, 6]
```

Iterating over Items in a List

- A useful thing we can do with lists is iterate over each element and perform an operation on those elements.
- For example, if we want to square each number:

```
a = [2, 4, 6]
for i in a:
    print(i ** 2)
```

```
4
16
36
```

Indentation Matters!

The following two code snippets only differ by indentation, but produce different output:

Snippet 1:

```
for i in a:
    print(i ** 2)
    print('hello')
```

Snippet 2:

```
for i in a:
    print(i ** 2)
print('hello')
```

`print('hello')` is part of the loop in case 1, and after the loop in case 2.

For Loops

If we want to save each squared number we could do:

```
a = [2, 4, 6]
b = []
for i in a:
  b.append(i ** 2)
b
```

```
[4, 16, 36]
```

List Comprehensions

However, an easier way to do this is to use *list comprehensions*:

```
a = [2, 4, 6]
b = [i ** 2 for i in a]
b
```

```
[4, 16, 36]
```

List Membership

We can check if a value is contained somewhere in a list using the `in` operator:

```
a = [2, 4, 6]
4 in a
```

```
True
```

```
5 in a
```

```
False
```

Copying Lists

Consider the following:

```
a = [2, 4, 6]
b = a
a[0] = 8
b
```

```
[8, 4, 6]
```

A change in a automatically changes b!

Copying Lists

To avoid this we can do:

```
a = [2, 4, 6]
b = a[:]
a[0] = 8
b
```

```
[2, 4, 6]
```

Two Types of Copying

- *Deep copy:* This copies a to b and recursively copies all of its elements, resulting in a completely independent object.
- *Shallow copy:* This copies a to b but does not recursively copy its elements. Instead it only copies the references to the elements in a (like the address for where in the computer's memory those elements are stored). This means that changes to elements of a will affect the elements of b.

Tuples

- A tuple is like a list with the important difference that they are *immutable*. Once we create them we cannot change them.
- We can create tuples with parentheses instead of square brackets:

```
a = (2, 4, 6)
```

- Indexing, iterating and other operations for lists also work with tuples.
- However, appending, popping, or altering values is not possible.

Tuple Assignment

Suppose we wanted to do the following:

```
x = ["a", "b", "c"]
x_0 = x[0]
x_1 = x[1]
x_2 = x[2]
```

We can shorten this code by using *tuple assignment:*

```
(x_0, x_1, x_2) = x
```