Eden Rusnell, Public Services Librarian @ Inforum on 2018-02-24 00:00:00
Workshop introduction
Why take this workshop?

• This workshop is designed with **absolute beginners** in mind
  • If you already know anything at all about programming, this workshop will probably be largely review for you

• This workshop is designed to teach you about what programming is and how to understand it
  • How programs work
  • How to understand process requirements
  • Learning to program on your own
Who am I?

• Eden Rusnell
• BA in History from McGill
• MI in ISD, also focusing on KMIM & KMD/UXD
• Currently working:
  • Inforum, Part-Time Public Services Librarian
  • Magnet/WhoPlusYou, Information Specialist (including database design and data integrity)
Workshop Goals

When the workshop is over, you will:
1. Know what programs can do and how
2. Understand basic data types and structures
3. Write basic functions in Python 3
4. Get resources for further learning of programming
Workshop Outline

1. Workshop introduction
2. What is programming?
3. Data types & structures
4. Designing programs
5. Programming with Python
6. Wrap up
What is Programming?
A brief history of programming

• The earliest computers were programmed in **binary**
  • Instructions were given to the computer as a series of 0 and 1

• **Paper tapes** and **punch cards** came after
  • These were patterns that were fed into mechanized looms
  • These were also written directly in machine code

• **Assembly languages** were first attempt to make programming more human-readable
  • These still used the architecture of machine language, but did not use the specific syntax

A brief history of programming

• The first widely-used programming language was **FORTRAN**
  • Short for **FORmula TRANslation**
  • These programs were **compiled** from **high-level programming languages**, which means they are translated from their written form into machine code

• Programming has moved to become more human readable
  • A program is an intermediary between your actions, or the actions you want to perform (which use a program's **interface**) and the actual computer
  • programming languages have to be translated into machine code

Source: Campbell-Kelly & Aspray, 1996
What is a programming language?

A programming language is a formalized language that can be translated into machine code that can give instructions to a computer. The code is then translated by a **compiler** or an **interpreter**:
- Interpreters do this translating line-by-line, as entered.
- Compilers do this to a whole document at once.

Source: Gaddis, 2012
Why learn Python?

Python is an open-source programming language that can accommodate both simple and complex processes.

- Python is considered to be one of the most flexible and forgiving programming languages
- Programming languages are classified by their paradigms: Python is a multiparadigm language, and falls into multiple classifications
  - These classifications include both compiled languages and interpreted languages

Program VS Interface

The front-facing element of any program is the interface. This is where user interaction takes place, so it's often called the user interface, or UI.

- The current model for user interfaces is the GUI, or graphical user interface, wherein you interact with computer elements in a visual way
  - For example, dragging a file into a different folder

- Prior to that, an operation like that would be performed with the terminal, or using the command line
  - move C:\Source Folder\File Name.doc C:\Destination Folder
Programming VS Markup Languages

A **markup language** is a language that is used to control the processing and presentation of text. The language can be used to specify formatting, layout, and style of information in a text file or on a web page.

- Markup languages have a far more limited vocabulary than programming languages
- HTML and XML are markup languages
Main elements of a program

The main elements of any program, large or small, can be understood as three steps: input, processing, and output.

1. The first step is to get **input**, which is any information or action put into a program
   - simple examples include keyboard input, text input, database entries
2. Next, the program will **process** the input
   - for a keyboard, this would be understanding the meaning of the keys pressed: for example, pressing the "s" key alone is different to holding the Shift key and pressing "s", which is different to pressing CTRL+S, which is different to CTRL+SHIFT+S
3. Finally, the program will produce **output**
   - In the case of keyboard input, this is respectively: writing "s" on the screen; writing "S" on the screen; saving the file; saving the file to a new document
Before you start programming...

DESIGN is the most important part of programming. Before you begin writing code, you want to make sure that you:

• Understand the task and its requirements
• Work from the problem, rather than working towards a solution

When coding, you want to focus on breaking down and understanding your task, rather than only focusing on a desired solution.
Task Breakdown

We're going to work together to break down a task into its component steps. Remember to consider the input-processing-output structure.

**Task Analysis**, of which this is a type, involves taking a task and breaking it out into its component steps, and any component steps for those steps, starting with a user goal

- A large portion of this is to help identify the **scope** of your activity

Source: Garrett, 2011
Task Example: Making Tea

1. Boil the water
   1. Fill the kettle
   2. Put the kettle heating element
   3. Wait for the water to heat up
   4. Take the kettle off the heating element
2. Put the tea into the tea pot
3. Pour the water into the tea pot
4. Wait for the tea to steep
5. Pour the tea into your cup & enjoy

Source: adapted from Garrett (2011)
Task Example: Making Tea

Defining what is **out-of-scope** is how you make decisions about what your program will do: for example, we are not including the steps of selecting your tea type, nor of putting loose-leaf tea into a tea bag.

2. Put the tea into the tea pot
   1. Select type of tea
      1. Open tea cupboard
      2. Pull out selected tea
   2. Get tea bag/tea strainer
   3. Put loose leaf tea into tea bag/strainer
Task Exercise: Getting to the iSchool

What are the steps involved in getting from your house to the iSchool
Task Exercise: Bringing an umbrella

On your own or in groups, identify the steps in the task of deciding whether or not you should bring an umbrella with you when you leave the house.
Our Example

On your own or in groups, identify the steps in the task of deciding whether or not you should bring an umbrella with you when you leave the house.
Pig Latin Example

Remember the Pig Latin program? Here's a breakdown of the steps to take a word and output it in pig latin:

• Get a word
• Check if the first letter is a vowel
• Check if the next letter is a vowel
  • Repeat until you find a vowel
• Return the appropriate pig latin version of the word
Data Types & Structures
Variables: Data Containers

A **variable** is a data container within your code: variables are created with assignment statements (referred to as "assigning a variable").

- **variable = expression**
  - The left-hand side (**variable** in the above example) creates the data container and gives it a label/name
  - the equal sign indicates that the value of the variable (**expression** in the example) is the right-side value

- A variable will take on a **data type** depending on the content of the expression
Data types

• strings
• integers
• floating points
• Boolean
String

A **string** is a sequence of characters that is used as data.

A **string delimiter** is a punctuation object that marks the start and end of your string.

- For Python, either single quotes ("string") or double quotes ("string") will work, as long as they are consistently applied ('don't do this')
- Strings that span multiple lines can be enclosed in triple quotes, like """"this string"""" or ""this string""""
Integers & floating point values

Integers
• Positive or negative whole numbers

example_variable = -25

Floating Point Values
• Sometimes called a float
• Positive or negative numbers with values to the right of the decimal place

example_variable = 1.99
Boolean

• These are values that can be either True or False.
  • This is a type of value that appears in many programming languages.

• You may already be familiar with Boolean Operators (AND, OR, NOT). Boolean operators follow those basic principles.
  • "Value One" AND "Value Two" – both values are True
  • "Value One" OR "Value Two" – either value is True, need not be both
  • "Value One" NOT "Value Two" – Value One is True AND Value Two is False
Logical Operators

- Boolean operators are also sometimes referred to as logical operators, and are used in evaluating True/False variables in sets.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a ∩ b</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a V b</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>¬a</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>
Data structures

We are also going to look at some simple data structures that can be used to contain multiple data elements.

• lists

• matrices (nested lists)
Lists

A list is a series of data elements

[100, 101, 102, 103]

A list does not have to have only one type of value:

["Eden", 25, 4.0]
Nested Lists or Matrix

You can have a set of lists within a list, which is called a matrix – you'll recognize this as a data table:

[[101, 102, 103],
 [104, 105, 106]]

Like lists, matrices are not limited to a single data type:

[['Name', 'Age', 'GPA'],
 ['Eden', 25, 4.0],
 ['Neo', 35, 1.0]]
Matrix in "table" format

This is the same matrix from the last slide, but formatted as a table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eden</td>
<td>25</td>
<td>4.0</td>
</tr>
<tr>
<td>Neo</td>
<td>35</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Designing Programs
Decision Structures

When you make a program, you will be instructing the computers on how to evaluate and respond to your input.

For our purposes, you will be outlining a series of tests for the computer to run against your input, and which are matched to instructions on what to output or how to react.

To model this behaviour, we use at decision structures.
Modeling a program

We use **flow charts** to model the decision structures within programs.
Should I wear a coat?

True
Wear a coat

Cold outside

False
Don't wear a coat
Qualifying for a loan

Make more than $30000

Don't qualify

More than 2 years in role

Don't qualify

Qualify
Design Exercise: Go to the movies

• Should I go to the movies tonight?
  • Do I already have plans to do something else?
  • Do I have to wake up early tomorrow?
  • Can I afford to go out to the movies?
  • Is there anything playing that I want to see?
Design Exercise: Bringing an umbrella

• Should I bring an umbrella with me today?
Pig latin example

1. Starting word
2. First letter is a vowel
   - True: Word + "yay" ("exampleyay")
   - False: Check next letter
3. Check next letter
   - False: Next letter is a vowel
     - False: Next letter is a vowel
       - True: Word + "ay" ("ordway")
"exampleyay"
Time for a break

We'll have a 15 minute break now. When we get back, we'll start programming.
Programming with Python
Starting resources

Before we get started, a couple of points:

• Writing code is **not** about memorization – when you program, you'll always be at a computer, so you can always look things up
  • it is not "cheating" to look things up when you are programming
• A good one to print up is Python's Beginner's Guide
  • There are two guides, one for Programmers and one for Non-Programmers
• Another good resource is the Python Language Reference
  • [https://docs.python.org/3/reference/index.html](https://docs.python.org/3/reference/index.html)
  • This is a guide to the core syntax of Python, and is a good resource if you're trying to remember how a particular command works in Python
Let's get coding

Open IDLE: you will be greeted with a small shell.

- IDLE stands for "Integrated Develoment Environment"
- Yes, in the same way Python is named after Monty Python, IDLE is named after Eric Idle

Note the ">>>" that appears on the right-hand side of the screen: that is a prompt for a new command.
Print

Let's try a command.

The way you structure a print command is:

```python
>>> print("Hello, world")
```

Note that the output appeared right away: IDLE is an interpreter.
Assigning a variable

The basic way to assign a variable is:

```python
>>> first_variable = "String value"
>>> second_variable = 15
>>> third_variable = 4.0
>>> fourth_variable = False
```

Can we identify the types of all these data types?
Rules for naming your variables

• Python has several **keywords** that cannot be used as variable names – see the table to the right
• variable labels can't contain spaces
• first character **must** be a letter or an underscore
  • subsequent characters must be letters, digits, or underscores

remember that variable names are **CaSe SeNsItiVe**.

Two popular styles: **snake_case** and **camelCase**
Rules for naming your variables

Which of the following variable names are illegal?

- hours_worked
- tax_at_13%
- employeeID
- 1st_item
- useR_id
- emailAddy
- allDaysWorked
- payroll_2018
- worker ID
- payroll_2018
- b00k_1d
- #_of_complaints
## Rules for naming your variables

Which of the following variable names are illegal?

<table>
<thead>
<tr>
<th>Illegal Variable Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>allDaysWorked</td>
</tr>
<tr>
<td>payroll_2018</td>
</tr>
<tr>
<td>worker ID</td>
</tr>
<tr>
<td>payroll_2018</td>
</tr>
<tr>
<td>b00k_1d</td>
</tr>
<tr>
<td>#_of_complaints</td>
</tr>
</tbody>
</table>
Printing a variable

When you tell the interpreter to print a variable, it will print the container's contents.

```python
>>> variable_name = "string variable"
>>> print(variable_name)
```

Let's try printing some of the variables we created.
Mathematical Operators

Addition +  
Subtraction -  
Multiplication *  
Division /  
Integer division //  
Remainder/"modulo" %  
Exponent **

Order of Operations (BEDMAS)
• Brackets
• Exponents
• Division and Multiplication
• Addition and Subtraction

The DM and AS of BEDMAS are done in left-to-right order.
Let's try some math

When writing mathematical equations, the convention is to leave spaces between each part of the equation, like "2 + 5" vs "2+5"

- Try assigning two numbers to two variables, then multiply them.
- Try assigning the result of that equation to a variable.
- Try multiplying a string.

```python
>>> "hip-hip-hooray!" * 3
```
## Relational Operators

<table>
<thead>
<tr>
<th>Boolean Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &gt; y</td>
<td>Is x greater than y?</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>Is x less than y?</td>
</tr>
<tr>
<td>x &gt;= y</td>
<td>Is x greater than or equal to y</td>
</tr>
<tr>
<td>x &lt;= y</td>
<td>Is x less than or equal to y?</td>
</tr>
<tr>
<td>x == y</td>
<td>Is x equal to y?</td>
</tr>
<tr>
<td>x != y</td>
<td>Is x not equal to y?</td>
</tr>
</tbody>
</table>
Numerical Ranges and Boundaries

• You can check if a value is outside or inside a range, as well as above or below a given value.
  • $x \geq 20$ and $x \leq 40$
  • $x < 20$ and $x > 40$

Assign an age number to a variable, then check if someone of that age is eligible to vote (18 and older)

>>> variable >= 18

>>> variable >= 18
Relational operators

• Find out if 18 is in range 22 to 47
• Find out if "Timmy" is greater than "Tommy"
• Find out if 123 is not equal to 123
ASCII codes & string comparison

• You can compare strings with "greater than" and "less than"
  • This will give you an answer based on the ASCII codes that make up each letter
  • You don’t need to know any more about ASCII codes unless you plan to work with them
• [https://ascii.cl/](https://ascii.cl/) provides a list of standard ASCII and Hex codes for common symbols (including letters)
Combining input

You can combine variables and print them with specific text.

```python
>>> room_number = "116"
>>> print("We are in room" + room_number)
We are in room116
```

Notice that the room number is a string: what happens if it's an integer?
Altering variables

You can also re-assign variables by adding onto them

```python
>>> variable = 12
>>> variable = variable + 1
```

You can also change the type of a variable: this is useful if you need to add an integer into a string that will be printed.

```python
>>> variable = str(variable)
```

`int()` converts a value to an integer, and `float()` converts a value to a float
## Augmented Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example Usage</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += 5</td>
<td>x = x + 5</td>
</tr>
<tr>
<td>-=</td>
<td>y -= 2</td>
<td>y = y - 2</td>
</tr>
<tr>
<td>*=</td>
<td>z *= 10</td>
<td>z = z * 10</td>
</tr>
<tr>
<td>/=</td>
<td>a /= b</td>
<td>a = a / b</td>
</tr>
<tr>
<td>%=</td>
<td>c %= 3</td>
<td>c = c % 3</td>
</tr>
</tbody>
</table>
Conditional Statements

When using programming to evaluate input, we can use a series of if-else statements. These are statements where if a condition is met, the program will output accordingly.

```python
>>> if animal == "dog":
    print("what a cute puppy!")
elif animal == "cat":
    print("what a cute kitty!")
else:
    print("you should think about adopting a pet")
```
Write a few if-elif-else statements

If a value is in a range, printing the value: this is equivalent to asking if value is >= [start of range] and value < [end of range].

```python
>>> value = 23
>>> if value in range(4,37):
    print(value)
23
>>> if value in range(4, 23):
    print(value)
>>>```

```python
```
Using nested if statements

You can do the same thing that you would do using ranges using a nested if statement. The two following statement will produce the same result as the statement using the range operator from the last slide.

A nested-if statement requires that both conditions be met

```python
>>> value = 23
>>> if value >= 4:
    if value < 37:
        print(value)

23
>>> 
```
Functions

A **function** is a way of grouping lines of code together.

- This is useful for using the same task in multiple places – you only have to write out a function once!
- This makes your code easier to read, and it avoid repeating code

- A function has the following parts:
  - a name
  - parameters or inputs
  - statements
  - return values or outputs (optional)
Example functions

This is a basic function which does not take any inputs

```python
>>> def hello_world():
    print("Hello, world")
```

When you call it by writing the function, it prints "Hello, world"

```python
>>> hello_world()
Hello, world
```
Functions with inputs

You can make the same function to take input this way:

```python
>>> statement = "Hello, world"
>>> def printing(phrase):
    print(phrase)

>>> printing(statement)
Hello, world
```
Creating functions

Create a function to print your name as a variable.

```python
>>> my_name = "Eden"
>>> def print_my_name(name):
    print(name)

In this example, the variable my_name is passed into the function print_my_name as an argument.

>>> print_my_name(my_name)
Eden
```
Creating functions

Create a function to print a personalized greeting (i.e., "hello" plus your name)

```python
>>> my_name = "Eden"
>>> def personalized_greeting(name):
    greeting = "Hello, " + name
    print(greeting)

>>> personalized_greeting(my_name)
Hello, Eden
```
Creating functions

Create a function to print your name if it starts with a letter in a range

```python
>>> my_name = "Eden"
>>> def printif(name):
    if name[0] in "AEIOU":
        print(name)

>>> printif(my_name)
Eden
```
Did you notice where it said "name[0]"?

The number in square brackets, "[0]", indicates that I wanted the function to compare only the item at the first index, or the first character in the string. This is different from comparing the whole string.

Indexes are also used to pinpoint entries in lists: this is something we can look at after we do the sample exercises.
Getting user input

You can also require user input throughout your function: this becomes its own input-processing-output cycle within your major input-processing-output cycle.

```python
>>> user_input = input("What is your name? ")
```
Creating a personalized greeting, again

Now we can use the input function of Python to create a personalized greeting that needs the user to enter their name.

```python
>>> name = input("What is your name?")
>>> personal_greeting = "Hello, " + str(name)
>>> print(personal_greeting)
```
Sample Exercises

1. Get the computer to say "Hello, world"
2. Create a personalized greeting
3. Create a variable with a room number and print the variable
4. Print a personalized greeting where you get the person's name and then tell them their room number
5. People need personalized receipts that tell them their subtotal, HST, and total for 2 items they want to purchase; they need to be able to enter each item cost separately
6. Samantha works 4 hours at day at the new minimum wage of $14.25; she wants to know how much she'll make in a 5-day work week before taxes (her gross pay)
This is a program in Python for Pig Latin

def pig_latinify(word):
    output = ""
    vowels = "aeiouAEIOU"
    notaword = " .,/?!"'()""
    word = str(word)
    if word.isdigit():
        raise ValueError
    if len(word) < 1:
        raise IndexError
    for letter in word:
        if word[0] in notaword:
            return "Make sure your input only contains letters"
        word = word[1:] + word[0]
    if word[0] in vowels:
        return word + "yay"
    for letter in word:
        word = word[1:] + word[0]
    if word[0] in vowels:
        return word + "ay"
    if vowels not in word:
        return word[1:] + word[0] + "ay"

print pig_latinify("word")
Wrap Up
Workshop Review

Today we covered:

• What is programming
• Breaking down tasks
• Creating decision structure flowcharts
• Basic data types and structures
• Basic python commands: assigning variables, printing, printing concatenated statements, changing variable types, reassigning variables, if-else statements, functions, getting user input
Resources

Python is an open-source programming language, so you can download it at [www.python.org](http://www.python.org). Their [Beginner's Guide](http://www.python.org) is a good place to learn more.

A tool that I like, which lets you walk through your code step-by-step, is [Python Tutor](http://www.python.org).

Lastly, [Stack Overflow](http://www.stackoverflow.com) is a great place to search for questions.
Tools

If you plan to do a lot of programming, I would recommend using an Integrated Development Environment (IDE)

**PyCharm Educational** is a free and open-source Python IDE

**GitHub** is a development platform that allows for change management and versioning, especially in projects involving multiple people or using multiple machines. It can be integrated with most IDEs.
(Some of) my sources


```python
>>> def any_questions():
    contact_info = "e.rusnell@utoronto.ca"
    user_input = input("Do you have any questions? (y/n): ")
    if user_input.lower() == "y":
        print("You can email me any questions at: " + contact_info)
    elif user_input.lower() == "n":
        print("Well, my email's " + contact_info + " if you think of any questions later")

>>> any_questions()
Do you have any questions? (y/n): Y
You can email me any questions at: e.rusnell@utoronto.ca

>>> quit()
```