ITEC 1150 Week 2 Chapter 1 Part 1

VARIABLES DATA TYPES, OPERATORS, FORMATTING, PROGRAM DEVELOPMENT PROCESS

Before We Begin…

A FEW HOUSEKEEPING ITEMS BEFORE JUMPING INTO THE FIRST LESSON

Weekly Content

- Remember, every week there will be:
  - Lecture notes and a lecture video – you are strongly encouraged to read the notes while watching the video. The two together are more complete than just one or the other.
  - Lab notes and a lab video – as with the lecture notes & video, you are encouraged to read and watch the lab content. The lab notes contain the exact requirements for the lab, while the video will provide more explanation and usually examples.

Communications

- Please add a photo to your D2L profile. The combination of name and photo helps me remember people better than just a name!
- If you don't check your school email regularly, consider forwarding it to your phone or a person email account. I'll be sending a weekly update to the whole class, as well as other more individual reminders and messages.
- You can send D2L notifications to your phone using the Pulse app.

Deep Dives

🤔

- When you see the 🤔 emoji on a slide, it means I'm explaining something in extra detail.
- These topics might be helpful to some people but might be confusing for others; it will depend on your learning style.
- I encourage you to read and think about what is on these slides, but <u>don't worry</u> if it doesn't entirely make sense.
- I will *not* expect you to know and understand what is on those slides; I'm including it just in case it helps someone understand a concept. If you don't need it or don't understand the extra detail, again, <u>don't worry!</u>

Practicing As You Go

- I strongly recommend that as you read the book and lecture notes and while watching the lecture and lab videos, that you code along in PyCharm.
- Every week, add a new directory (folder) to your existing project, and then add a Python file to that directory named something like "exercises.py". Use that file to work on examples, exercises, and experiment on your own.
- There's no better way to learn than by predicting how something will work and trying it out. You either find out you were right or that there's something you didn't know that you now get to learn!

Programming Syntax

- Syntax is the name for the rules you must follow when writing programs
- Syntax is like grammar in human languages, but programming syntax is *very strict*, more so than human grammar. (A human will 'get what you mean' even if your grammar is off; a computer cannot.)
- If PyCharm can tell you made a mistake, it will tell you and give you hints, but sometimes you won't know about a mistake until your program crashes.
- A mistake can be using the language incorrectly, or it could be a single space, a typo, or capital letter that is causing a syntax error.
- More on this later…

Conventions

- **Conventions** are rules that we follow that aren't strictly necessary, but they make things easier
  - Saying 'please' and 'thank you' or taking turns are examples of *social conventions.*
- Programming has conventions as well and are concerned with details of how we write programs, name variables and functions, and so forth. A couple we will follow in this class (the book may have differences, but follow these):
  - All files begin with a header in the format we used in the Week 1 lab.
  - Words in variable names are separated with underscores, like this:
    - `my_name, hours_worked, total_pay`
  - That way of naming is called **snake_case** and is a Python convention. (Another common convention is **camelCase**, which is used in Java, among others.)

Conventions can be common to languages but can be more specifically agreed on by a workplace or project team.

Agenda For This Week

- VARIABLES
- DATA TYPES
- OPERATORS
- FORMATTING
- PROGRAM DEVELOPMENT PROCESS

WHERE WE STORE DATA IN OUR PROGRAM

Variables

What's a Variable?

Try this out yourself!

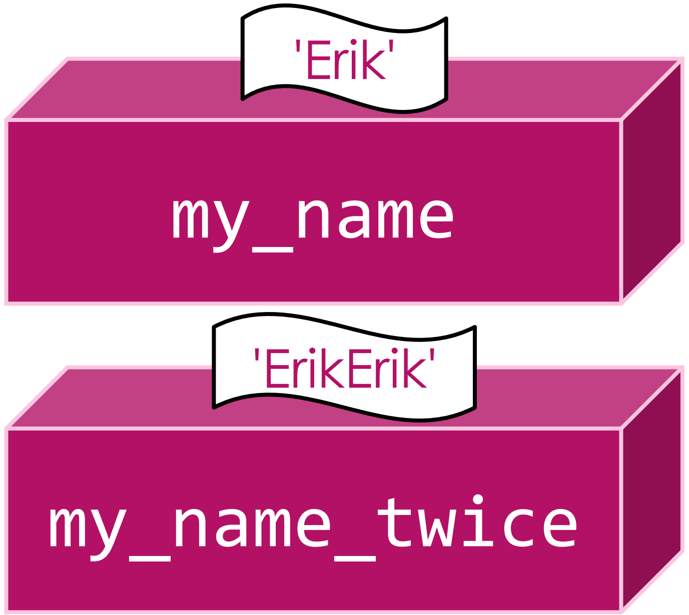
Variables are holding places in memory where a program stores data.

To create a variable, just give it a descriptive name and assign a value to it, like this:

```
my_name = 'Erik'
my_name_twice = my_name + my_name
```

`my_name` is the variable, and the value is 'Erik'

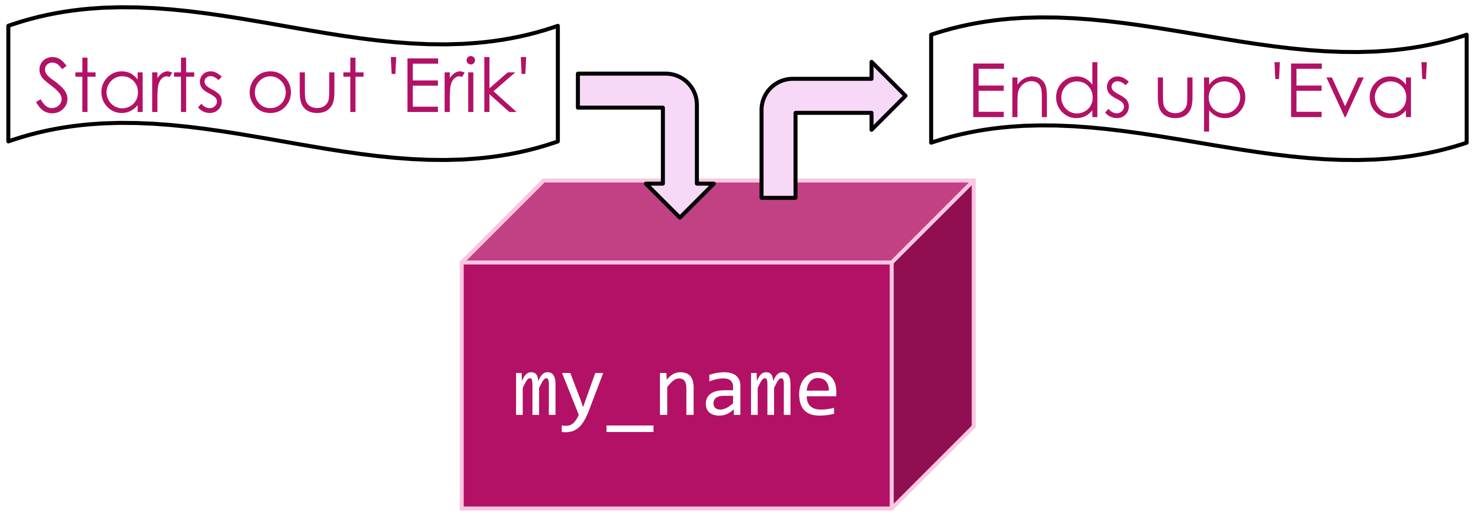
`my_name_twice` is another variable; do you understand what the value is?



Variables (cont.)

- You can create variables, store data in variables, fetch data from variables, and change data in variables.

```
my_name = 'Erik'
print(my_name)
my_name = 'Eva'
print(my_name)
```



Variables (cont.)

- The = symbol in Python is known as the **assignment operator**
  - On the left side of the = sign is the variable name
  - On the right side of the = sign is the value stored under that name
- You can change the value assigned to a variable any time you like
- Refer to the variable value by its name

```
my_college = 'MC'
print(my_college) # NOT print('MC')
```

- Dynamic variables work - for data that changes!

```
my_college = 'Metro State'  # future plan
print(my_college)
```

Rules For Variables

- When you create a variable, you must assign something to it at the same time.
- Variable names in Python can be any length and can consist of:
  - uppercase and lowercase letters (A-Z, a-z)
  - digits (0-9)
  - and the underscore character (_)
- A variable name cannot *start* with a digit.
- Variable names are case-sensitive:
  - `my_name` and `My_Name` are different variables!
  - It's best to stick to all lower-case to avoid getting confused.

A CONCEPT THAT IS BOTH SIMPLE AND A LITTLE CONFUSING

Data Types

Why Data Has Types 🤔

Deep dive alert!

- As you've probably heard, computers store data in binary, meaning as long strings of ones and zeroes.
- Binary numbers (also called base-2) work just like regular numbers (called base-10), except that instead of the digits 0-9, they only use the digits 1 and zero.
- So, computers can store any number, they just do it in base-2 instead of base-10.

127 in binary is 01111111

We can do addition or multiplication with a binary number the same way we do with a binary number the same way we do with a base-10 number:

```
  1111111
+ 1111111
---------
 11111110, which is 254 in base-10.
```

Why Data Has Types (cont.) 🤔

- However, storing something like "Hello World!" is more of a challenge, because letters and punctuation aren't numbers.
- To solve this problem, programmers have developed standards that say, "This binary number means that character."
- We need to give types to our data because the computer needs to know if 01001000 should be treated as a number or if it should represent the capital letter H.

ASCII and Unicode are nearly universal standards that spell out this conversion. Here's **Hello World!** in binary using ASCII:

01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010 01101100 01100100 00100001

Each 8-digit number refers to a specific character in the ASCII table.

See https://www.ascii-code.com/ for reference.

Our First Data Types

- This week, we'll be looking at three types of data, and some of the things Python helps us do with data of each type.
- The types are:
  - Integers, which are whole numbers
  - Floating point numbers, which are numbers with a decimal point
  - Strings, which are lists of characters (letters, punctuation, spaces, tabs, and even the digits 0-9.)

Data Types: Strings

- The variables `my_name` and `my_college` from a few slides ago contain **string** data. The name "String" comes from the idea that text is just a string of **characters**.
- Characters are:
  - Capital and lowercase letters
  - Punctuation marks
  - The digits 0-9 (if we're using them as text instead of as numbers)
  - Non-printing characters, like tabs, spaces, "new line", etc.
  - Even emoji are characters
- Any set of characters enclosed with quotes is a string:

```
'Erik'
'Minneapolis College'
'Python is cool'
'@!@#@#$#%$^^&^&**&)(%%$SDFDXB'
'1234'
```

Yup, still a string! To the computer, the characters '1234' are nothing like the integer 1234.

Quotation Marks Around Strings

- You can use either double quotes or single quotes, but make sure they match.
- OK:

```
"Fall 2018"
'Cookies'
'123456789'
```

- Not OK:

```
'Hello World"
"This class is ITEC 1150'
```

Quotation Marks *In* Strings

- Since quotes are used to indicate the beginning and end of a string literal, what do you do if your string has a quote or apostrophe in it?
- If you try to run this code - what happens? Why?

```
quotation = "She said, "I like coding," to me."

store = 'Macy's'
```

Quotation Marks *In* Strings

- One way to fix this is to use one type of quotation mark around the string, and the other type inside the string:

```
quotation = 'She said, "I like coding," to me.'
store = "Macy's"
print(quotation)
print(store)
```

However, this is easy to get wrong, and there are situations where it just won't work.

A better approach is to learn to use **escape characters**.

The Escape Character

- The escape character is a single backslash: \. It tells Python that you want the next character to be part of the string:

`\'`	lets you put a single quote into a string
`\"`	lets you put a double quote into a string

- The escape character can also be used to put non-printing characters into a string:

`\n`	newline
`\t`	tab

- If you need a backslash \ as a character in a string literal, you will need to enter the escape character first.

`\\`	to see 1 backslash in a string literal

Escape Characters (cont.)

- What do you think these will print? Predict, then try them out!

```
print('One \nTwo \nThree')
print("She said, \"I'm learning Python,\" to me.")
print("Here is a tab\t Did you see it?")
```

Data Types: Integers

- What about numbers?
- We can't do math with strings, even if it's the string '123'
- Integer variables store whole numbers:

```
temp_in_january = -6
number_of_students = 87
```

- Notice there are no quotes around the values
- And no symbols like $ or F (that's formatting)
- With variables of this type, you can do math – let's experiment…

How integers and strings are different:

```
print(123 + 123)
>>> 246

Print('123' + '123')
>>> '123123'
```

Integers (cont.)

- Here is a simple program using integer data types

```
number_of_cds = 45
number_of_lps = 23
number_of_albums = number_of_cds + number_of_lps
print(number_of_albums)
```

Data Types: Floating Point Numbers

- Decimal places are hard for a computer to handle, because a number like 0.333 repeating (1/3) goes on forever and the computer has a limited amount of memory.
- Computers use a special method of dealing with decimal numbers, where they use a fixed amount of memory and the decimal point 'floats' around depending on how many digits are needed on each side of the decimal point.
- We call this data type a **Float**, which means any number with a decimal point:

```
# Python IDs these as a float variables, even if it’s .0

temp_today = 35.0

coffee_price = 3.45
```

Identifying The Type of a Variable

- The data in every variable has a type
- Python figures out the variable type from the data entered

```
# here are two variables, of different types
college = 'MC'  # string data entered
number_of_cds = 3   # integer data entered
```

- We can run the `type()` function against the variables, to see Python's interpretation. And you must add the `print()` function, to see the answer:

```
# nested functions
print(type(college)) # adapt to see the type of the other variable
```

WHAT THEY DO

LIMITATIONS

Operators

Math Operators

- Many math operators are things you already know, like addition, subtraction, multiplication, and division.
- Floor division gives you the *quotient*, but not the remainder.
- Modulo gives you the *remainder*, but not the quotient.

Operator	Description	Example	Result
+	Addition	2 + 3	5
-	Subtraction	5 – 3	2
*	Multiplication	2 * 3	6
/	Division	11 / 4	2.75
//	Floor division	11 // 4	2
%	Modulo	11 % 4	3
**	Exponentiation	2 ** 3	8

String Operators

- There are fewer operators that work with strings
- *Concatenation* just means 'joining together'

- Operators like division don't make sense for strings and will give an error message

Operator	Description	Example	Result
+	Concatenation	'Hello' + 'World'	'HelloWorld'
*	Repetition	'AB' * 3	'ABABAB'

Operators And Mixed Types

- Strings and numeric types cannot be concatenated. One way to join them is to use a function to convert your number to a string (more on this in the next section):

```
'Number of students = ' + str(30)
```

- Integers and floats can be mixed and matched with the math operators. If a float is used with an operator, the result is always a float:

```
result = 10 / 2.0
print(type(result))

>>> <class 'float'>
```

Formatting Output

TODAY WE'LL ONLY LOOK AT ROUNDING NUMBERS, JOINING STRINGS, AND CONVERTING NUMBERS TO STRINGS.

WE'LL COVER FORMATTING IN MORE DETAIL IN THE COMING WEEKS.

Turning Data Into Information

- You often need to mix strings and numeric data (integers or floats) in output sentences. Here are two basic ways to do this:

```
total = 999.99
print('Total wages = $' + str(total))
print('Total wages = $', total)
```

The str() function turns a number into a string.

This form of the print() function automatically turns values into strings. (It calls str() itself inside the function—we'll learn more about functions later.)

- Note: the two forms of the print statement have a difference in their output as well – can you see it? You need to make sure to use a method that gives the output you want.

Rounding Numbers

- The `round(number, places)` function can be used to get a new, rounded number:

```
float_number = 12345.6789
float_number_rounded = round(float_number, 2)
print('Your number is', float_number_rounded)
```

- Important: if you do a follow-up test with `print(float_number)`, you see the original value is not changed by the `round()` function—it returns a *new* value while leaving the original alone.

Format Strings

- More advanced formatting can be done with **formatting strings**. Look carefully at the fourth line:

```
float_number = 12345.6789
float_number_rounded = round(float_number, 2)
print('Your number is', float_number_rounded) # easy! But no thousands separator
print(f'Your number is {float_number:,.2f}') # format code adds thousands separator & makes it look like it's rounded
```

- It's best to do both—round the variable calculation if desired, and then use it with a format code (the 2nd print statement). This will ensure things like a trailing zero will show up.
- This topic will be detailed in the next lesson.

Debugging And Testing

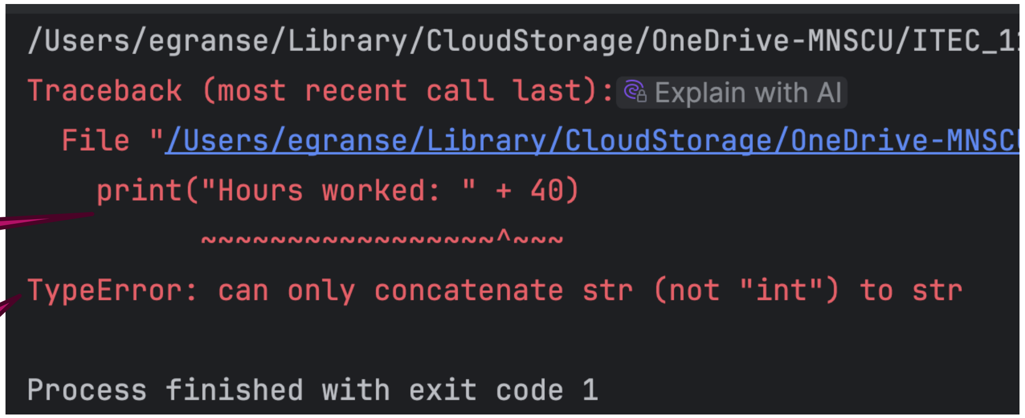
YOU WILL SEE ERRORS

HOW TO UNDERSTAND AND FIX THEM

HOW TO CHECK YOUR PROGRAM FOR CORRECTNESS

Example Error Message

- Error messages in programming are often intimidating because they're very technical. Do your best to focus on what you understand and ask for help if you're stuck.



```
/Users/egranse/Library/CloudStorage/OneDrive-MNSCU/ITEC_1
Traceback (most recent call last):
  File "/Users/egranse/Library/CloudStorage/OneDrive-MNSCU
    print("Hours worked: " + 40)
          ~~~~~~~~~~~~~~~~~^~~~
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```

This shows us *where* the problem is

This shows us *what* the problem is (can you guess what I did wrong?)

Testing

- Always run your program to make sure it does what you expect.
- As weeks go by and our programs get more complex, test with both good and bad data—programmers often only test the way the already know it works and forget that errors and bad input happen! This causes bugs in real-world programs all the time.