Introduction to Computational Thinking and Python Programming

Sarom Leang, Ph.D. (Instructor)

Jesse McCandlish (Mentor)

March 25, 2023

Session 4

[REMINDER]

Names and Faces and Pronouns

Sarom Leang, Ph.D. (Instructor)

- Professor
- Instructor
- Mr. Leang

Jesse McCandlish (Mentor)

- Jesse

[REMINDER]

Schedule

10:00 AM – 10:15 AM **Homeroom**

10:15 AM – 11:35 AM **G1 Block**

11:35 AM – 12:35 PM **Break/Lunch**

12:40 PM – 02:00 PM **G2 Block**

80 minutes of class time

[REMINDER]

Student Expectations

- **NO FOOD**
- **NO DRINKS** (on the table)
- Be respectful to individuals and property
- Be open to learning
- Be open to not understanding
- Be patient with yourself
- Ask questions
- Explore
- **Embrace failure**

[UPDATED]

Lesson Plan

Session #1 – October 29, 2022

- ~~Entrance survey~~
- Python programming environment, “Hello World”

Session #2 – December 3, 2022

- ~~Parallel computing~~
- ~~Python programming environment, “Hello World”~~
- Python arithmetic operators, integer, float, and string data type

Session #3 – March 4, 2023

- Generative AI
- Python arithmetic operators
- ~~Python Booleans data type, comparison and logical operators, truth tables~~

Session #4 – March 25, 2023

- Python integer, float, string, Booleans data type, comparison and logical operators, truth tables
- Python control structures, and collections {list, dictionary, tuple}

Session #5 – April 22, 2023

Technology in the News

- **TikTok**
  - Maybe be banned from the US
  - Data privacy for data in flight and at rest
  - Data in flight (transmitted) maybe subjected to laws governing the data centers that the data is routed through
  - Data at rest (stored) is subjected to laws governing the data center that the data is stored in
- **Generative AI** (still)
  - **GPT-4** – a new large language model that is multimodal (can accept text and image inputs) and output generative text

[SESSION 3]

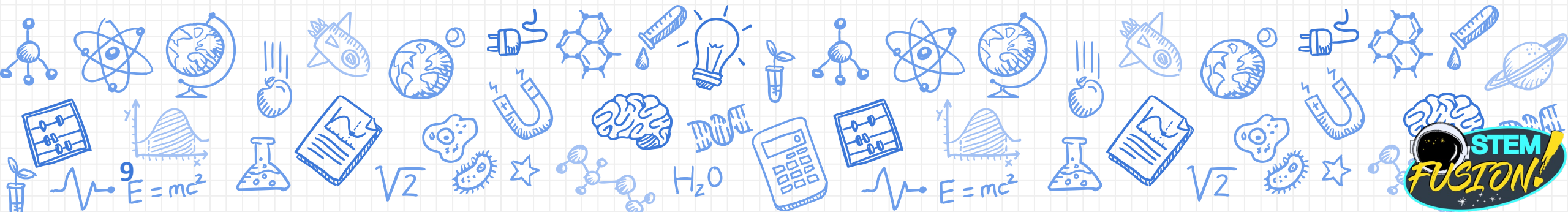
Generative AI and Academic Code of Conduct

- Cheating or violating academic code of conduct using generative AI is unethical and has serious consequences.
- Disciplinary action may include failing the assignment or course, or even expulsion from school.
- Long-term consequences may include damaging a student's reputation and future academic and career prospects.
- **Use generative AI responsibly and in accordance with the academic code of conduct.**

Generative AI DALL-E Competition

- **Entry**
  - **Email: ssok1@gmu.edu**
  - **Subject: DALL-E Submission**
  - **Body of email:**
    - **Full name**
    - **Session**
    - **DALL-E prompt used**
  - **Attachment: One (1) DALL-E generated image**
  - **Entry cutoff time: 2PM**

Hello World



Programming Environment

- A text editor
  - Sublime https://www.sublimetext.com
- A Python interpreter
  - Python 3.11.0 https://www.python.org

First Programming Exercise: Hello World

- Open up a blank document in the Sublime editor
- Type: `print("Hello World!")`
- Save the file as **hello.py** into your **Documents** folder
- Open up the command prompt
  1. Test if python is working: type `python` and hit enter
  2. If you get an error then: type
     `set PATH=%PATH%;C:\Python311`
     and hit enter
  3. Repeat step 1
- Execute your script:
  - Switch into the directory containing **hello.py**
    - `cd Documents`
  - Call the python program to interpret and execute your code
    - `python hello.py`

Arithmetic Operators in Python

Operator	Usage	Operation
+	x + y	Addition
-	x - y	Subtraction
*	x * y	Multiplication
/	x / y	Division
%	x % y	Remainder
**	x ** y	Exponentiation
//	x // y	Floor division

Operator Precedence

- In Python, operators will be evaluated in order of precedence.
- Order of operations – **PEMDAS**
  1. **P**arentheses ( )
  2. **E**xponentiation **
  3. **M**ultiplication * and **D**ivision / , // , %
  4. **A**ddition + and **S**ubtraction –
- After **PEMDAS**, order goes left to right.
- Use parentheses to override order.

Class Exercise: PEMDAS

Try to determine the solution by hand and then verify with Python

```
5 + (4 - 2) * 2 + 4 % 2 - 4 // 3 - (5 - 3) / 1 = ?
```

Class Exercise: PEMDAS

```
5 + (4 - 2) * 2 + 4 % 2 - 4 // 3 - (5 - 3) / 1 = ?
5 + (4 - 2) * 2 + 4 % 2 - 4 // 3 - (5 - 3) / 1 = ?
5 + 2 * 2 + 4 % 2 - 4 // 3 - 2 / 1 = ?
5 + 2 * 2 + 4 % 2 - 4 // 3 - 2 / 1 = ?
5 + 4 + 0 - 1 - 2 = 6
```

Data Types in Programming

- Different categories of information that we can **use** and **manipulate** in a program.
- We manipulate these data types by performing an **operation** (or applying a **method**).

Information	Operations			
	+	–	*	/
Numbers	Addition	Subtraction	Multiplication	Division
Words	Concatenation	?	?	?

Basic Data Types in Python

- **Integer**
- **Floating-point**
- **String**
- Boolean
- List
- Tuple
- Set
- Dictionary

Integer

In Python, any number **without** a decimal is considered an integer:

```
 0
 1
 100
 10000000000000000000000000001
-10000000000000000000000000001
-100
-1
```

Floating-point

In Python, any number **with** a decimal is considered an floating-point:

```
 1.0
 100.0
 10000000000000000000000000001.0
-10000000000000000000000000001.0
-100.0
-1.0
```

Class Exercise: Integer or Float

- What data type would you use?
  - The number of people that came on your fishing trip
  - Length of a fish you caught, in meters
  - Number of fish caught on a fishing trip
  - Length of time it took to catch the first fish, in hours
  - The amount of fuel to fill up the fishing boat, in gallons
  - The total cost of fuel to fill up the fishing boat, in dollars
  - The number of slices in a pizza
  - The number of holidays in a calendar year
  - The pH level of a swimming pool
  - The average age in your high school class
  - The number of hours you worked in a day

Python `type()` function

Returns the type of the argument passed as a parameter.

```
print(type(1))
print(type(1.0))
```

Programming Exercise: Maths

Operation	Result	Operation	Result
Let x = 4 and y = 2		Let x = 4.0 and y = 2	
`print(type(x + y))`		`print(type(x + y))`	
`print(type(x – y))`		`print(type(x – y))`	
`print(type(-x))`		`print(type(-x))`	
`print(type(x*y))`		`print(type(x*y))`	
`print(type(x/y))`		`print(type(x/y))`	
`print(type(x//y))`		`print(type(x//y))`	
`print(type(x%y))`		`print(type(x%y))`	
`print(type(x**y))`		`print(type(x**y))`	

String

In Python, a string is a sequence (array) of characters.

```
print("Early Identification Program")
print('Early Identification Program')
print("""Early Identification Program""")
print("""Early
Identification
Program""")
```

String Methods

Method	Result
String.capitalize()	Converts first character to capital letter
String.upper()	Returns uppercased string
String.lower()	Returns lowercased string
String.swapcase()	Returns string with casing swapped
String.count(*substring*)	Returns occurrences of substring within string
String.replace(*old, new, count*)	Replace occurrences of *old* with *new* in string

String Methods

- `"Early Identification Program".capitalize()`
- `"Early Identification Program".upper()`
- `"Early Identification Program".lower()`
- `"Early Identification Program".swapcase()`
- `"Early Identification Program".count("ti")`
- `"Early Identification Program".replace("ti","",1)`
- `"Early Identification Program".replace("ti","",2)`
- `"Early Identification Program".replace("ti","")`

String Operations

- `"Early" + "Identification" + "Program"`
- `"Early" - "Identification" - "Program"`
- `"Early" * "Identification" * "Program"`
- `"Early" / "Identification" / "Program"`
- `"Early" * "5"`
- `"Early" *  5`

Type Casting (Type Conversion)

```
print(float(1))
print(int(1.0))
print(str(1.0))
print(str(1)+1)
print(str(1)+'1')
print(int(1.0)+'1')
print(int(1.0)+int('1'))
```

Type Casting (Type Conversion)

- To typecast to a float: `float()`
- To typecast to an integer: `int()`
- To typecast to a string: `str()`

Basic Data Types in Python

- Integer
- Floating-point
- String
- **Boolean**
- **List**
- **Tuple**
- **Set**
- **Dictionary**

Boolean

- A data type with only two possible values (`True`, `False`) used for logic.
- Typecast using `bool()`

Boolean in Python

```
print(True)
print(False)
print(bool(0))
print(bool(1))
print(bool(1123.23))
print(bool(-500))
print(bool("a"))
```

Comparison Operators in Python

Operation	Description
x == y	**True** if the value of x is **equal** to the value of y
x != y	**True** if the value of x is **not equal** to the value of y
x <> y	**True** if the value of x is **not equal** to the value of y
x > y	**True** if the value of x is **greater than** the value of y
x < y	**True** if the value of x is **less than** the value of y
x >= y	**True** if the value of x is **greater than or equal to** the value of y
x <= y	**True** if the value of x is **less than or equal to** the value of y

List

- A collection of items in a particular **order**. Items (or **elements**) are contained within square brackets with items separated with a comma.

```
[]
["milk","bread"]
[1.6180339887,3.1415926535]
["milk",2.0,300,True,False]
```

List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list, to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

List versus Array

- Can anyone tell me the difference between a list and an array?

List versus Array

- A **list** can hold elements of **different** data types (e.g., int, float, string, Boolean).
- An **array** can only hold elements of the **same** data type.
- Arrays are faster and more memory efficient when performing operations. (Why?)
- Fun fact: Arrays are not built into the Python language. You must import additional libraries to use them.

List/Array Index

- An **index** is used to define the location of an element.
- In most modern programming languages index will start at **0** (referred to as zero-base indexing/numbering)
  - C, C++, Python, Java
- Older programming languages (and a few newer languages) will start at **1** (referred to as one-based indexing/numbering)
  - Fortran, Julia, MATHLAB, R, Wolfram

```
Index:    0       1     2      3      4
        ["mop",2.000,300,True,False]
```

Accessing Elements in an List/Array

```
print(["mop",2.000,300,True,False][0])
print(["mop",2.000,300,True,False][1])
print(["mop",2.000,300,True,False][2])
print(["mop",2.000,300,True,False][3])
print(["mop",2.000,300,True,False][4])
print(["mop",2.000,300,True,False][5])
print(["mop",2.000,300,True,False][-1])
print(["mop",2.000,300,True,False][-2])
print(["mop",2.000,300,True,False][-3])
```

Accessing Elements in List/Array

```
print(["mop",2.000,300,True,False][0])  = "mop"
print(["mop",2.000,300,True,False][1])  = 2.0
print(["mop",2.000,300,True,False][2])  = 300
print(["mop",2.000,300,True,False][3])  = True
print(["mop",2.000,300,True,False][4])  = False
print(["mop",2.000,300,True,False][5])  = Error
print(["mop",2.000,300,True,False][-1]) = False
print(["mop",2.000,300,True,False][-2]) = True
print(["mop",2.000,300,True,False][-3]) = 300
```

Tuple

- Similar to a list but they are **immutable** (values cannot be changed once they are created).

```
()
("milk","bread")
(1.6180339887,3.1415926535)
("milk",2.0,300)
```

Tuple Methods?

- Like a list but immutable.

Tuple Methods?

Method	Description
~~append()~~	~~Adds an element at the end of the list~~
~~clear()~~	~~Removes all the elements from the list~~
count()	Returns the number of elements with the specified value
~~extend()~~	~~Add the elements of a list, to the end of the current list~~
index()	Returns the index of the first element with the specified value
~~insert()~~	~~Adds an element at the specified position~~
~~remove()~~	~~Removes the first item with the specified value~~
~~reverse()~~	~~Reverses the order of the list~~
~~sort()~~	~~Sorts the list~~

Accessing Elements in Tuple

```
print(("mop",2.000,300,True,False)[0])  = "mop"
print(("mop",2.000,300,True,False)[1])  = 2.0
print(("mop",2.000,300,True,False)[2])  = 300
print(("mop",2.000,300,True,False)[3])  = True
print(("mop",2.000,300,True,False)[4])  = False
print(("mop",2.000,300,True,False)[5])  = Error
print(("mop",2.000,300,True,False)[-1]) = False
print(("mop",2.000,300,True,False)[-2]) = True
print(("mop",2.000,300,True,False)[-3]) = 300
```

Accessing a Value in a Dictionary

```
{"color":"red","is heavy":True,"cost":4.99}["color"]
{"color":"red","is heavy":True,"cost":4.99}["is heavy"]
{"color":"red","is heavy":True,"cost":4.99}["cost"]
{"color":"red","is heavy":True,"cost":4.99}[0]
```