INTRO TO PROGRAMMING WITH PYTHON

FUNDAMENTALS: VARIABLES + OPERATORS

Wednesday, February 12 2025 TOH210

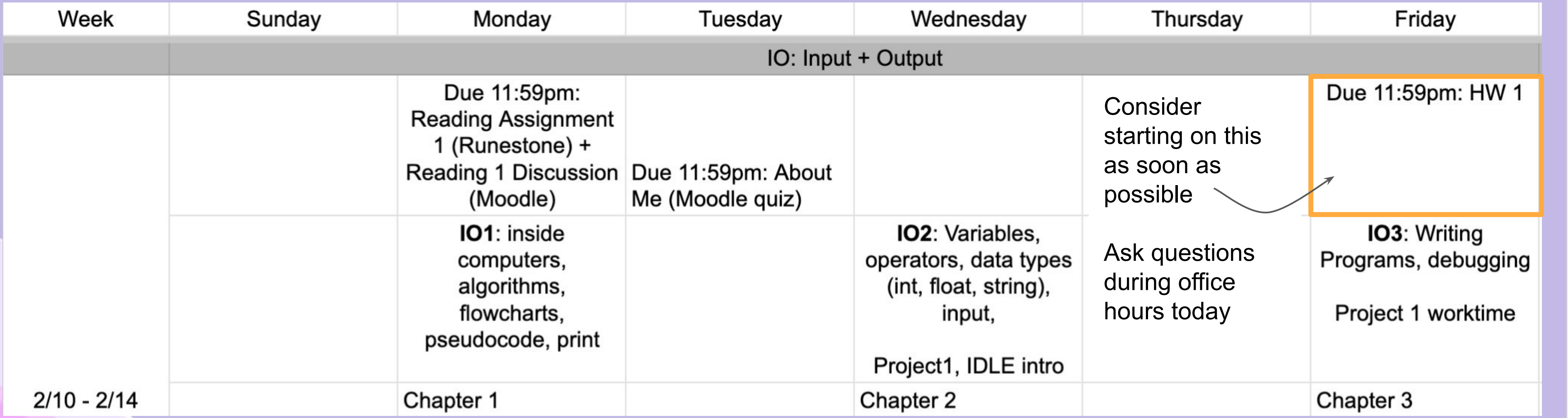
TAKEAWAYS FROM LAST TIME

Things you should know:

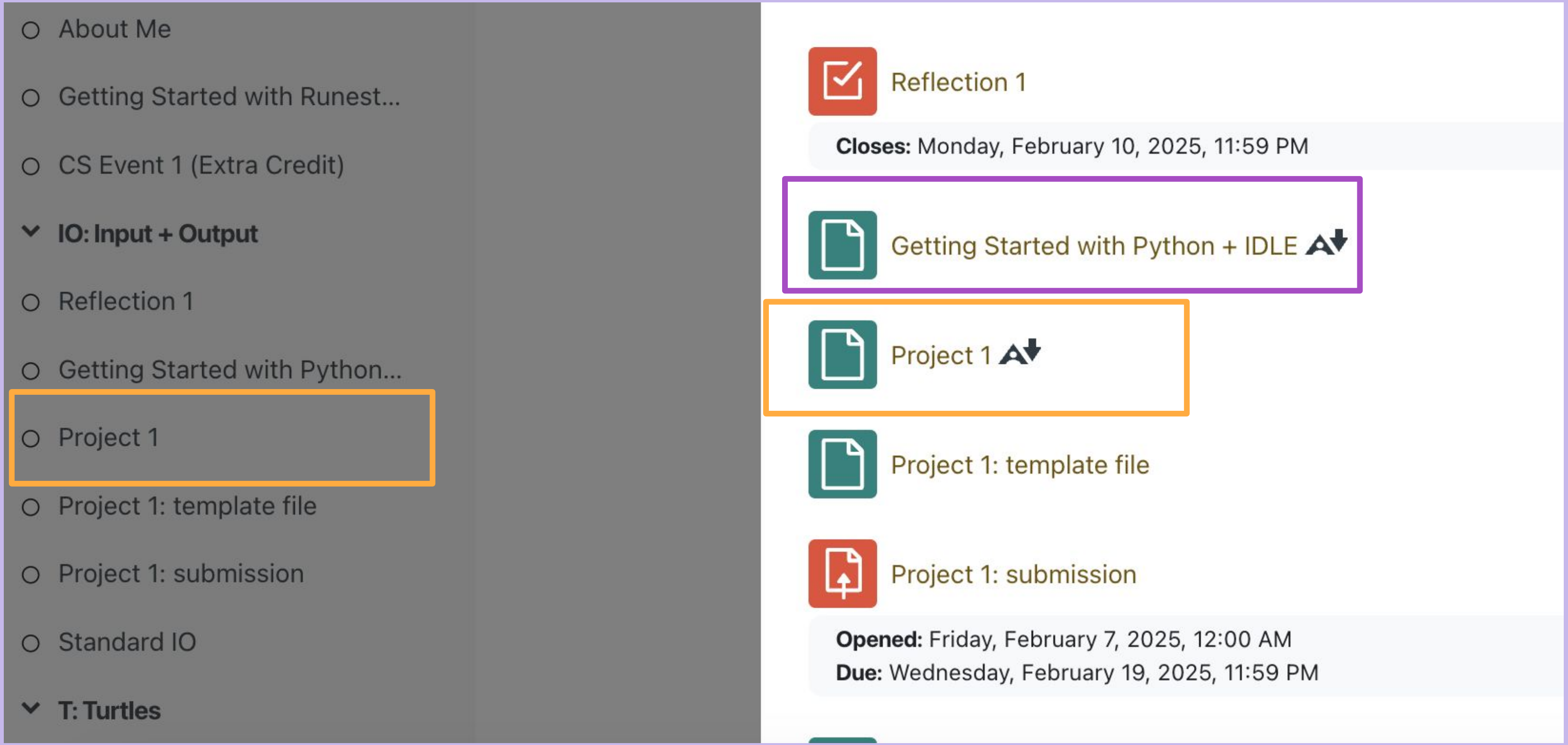
- what an algorithm is, how to think algorithmically
- pseudocode
- computers talk in binary
- we use Python

UPCOMING SCHEDULE

google sheet link



Week	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday
	IO: Input + Output					
2/10 - 2/14		Due 11:59pm: Reading Assignment 1 (Runestone) + Reading 1 Discussion (Moodle)	Due 11:59pm: About Me (Moodle quiz)		Consider starting on this as soon as possible	Due 11:59pm: HW 1
		IO1: inside computers, algorithms, flowcharts, pseudocode, print		**IO2**: Variables, operators, data types (int, float, string), input, Project1, IDLE intro	Ask questions during office hours today	**IO3**: Writing Programs, debugging Project 1 worktime
		Chapter 1		Chapter 2		Chapter 3



- ○ About Me
- ○ Getting Started with Runest...
- ○ CS Event 1 (Extra Credit)

⌄ **IO: Input + Output**

- ○ Reflection 1
- ○ Getting Started with Python...
- ○ Project 1
- ○ Project 1: template file
- ○ Project 1: submission
- ○ Standard IO

⌄ **T: Turtles**

Reflection 1

Closes: Monday, February 10, 2025, 11:59 PM

Getting Started with Python + IDLE

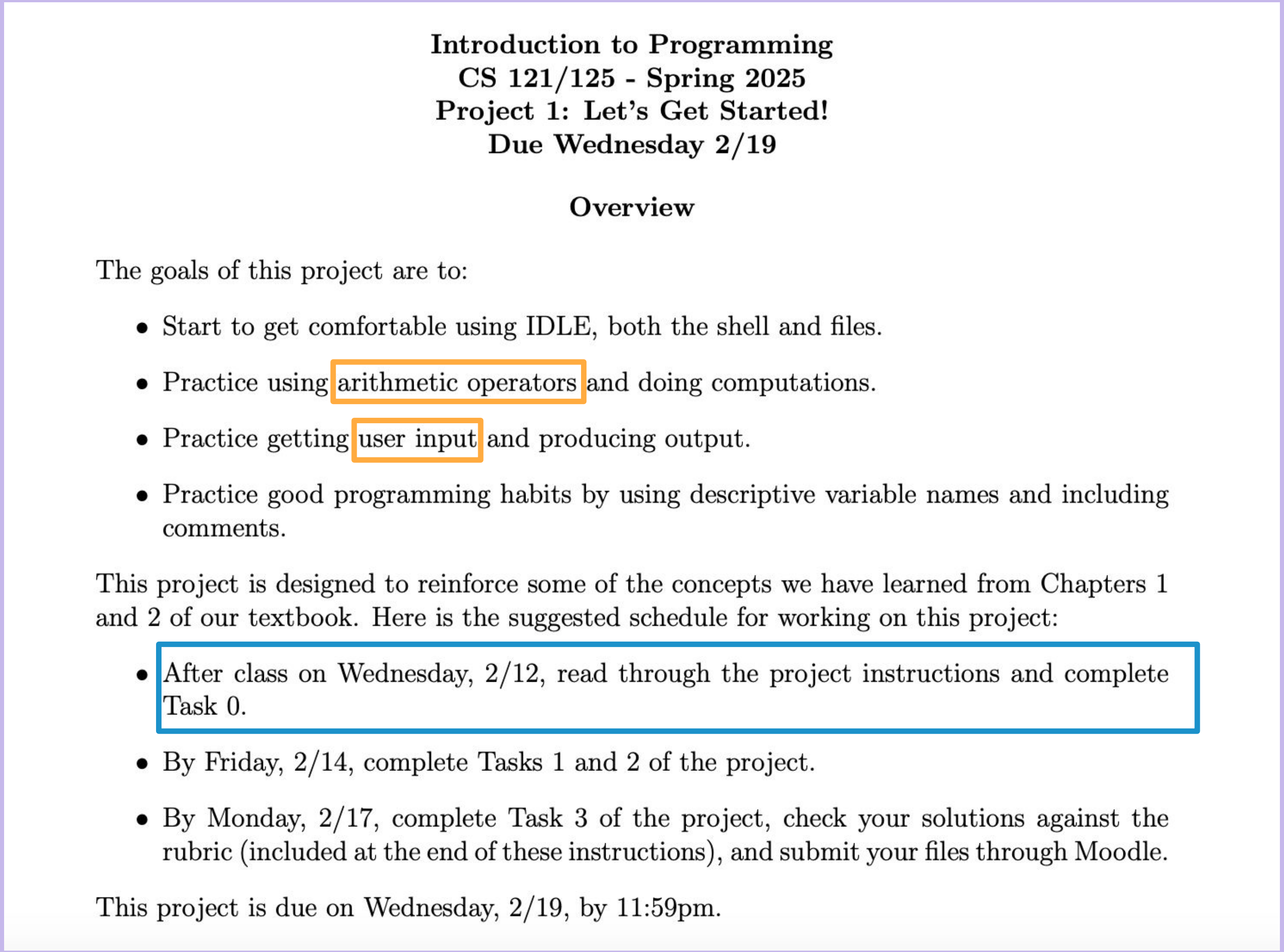
Project 1

Project 1: template file

Project 1: submission

Opened: Friday, February 7, 2025, 12:00 AM
Due: Wednesday, February 19, 2025, 11:59 PM

Project 1 and IDLE+Python guide both up on Moodle!



Introduction to Programming
CS 121/125 - Spring 2025
Project 1: Let's Get Started!
Due Wednesday 2/19

Overview

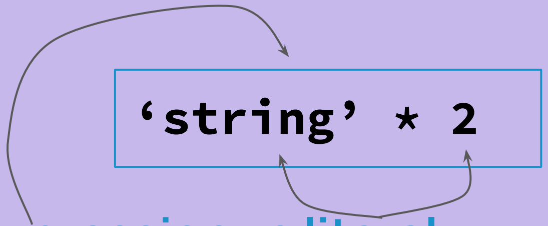
The goals of this project are to:

- Start to get comfortable using IDLE, both the shell and files.
- Practice using arithmetic operators and doing computations.
- Practice getting user input and producing output.
- Practice good programming habits by using descriptive variable names and including comments.

This project is designed to reinforce some of the concepts we have learned from Chapters 1 and 2 of our textbook. Here is the suggested schedule for working on this project:

- After class on Wednesday, 2/12, read through the project instructions and complete Task 0.
- By Friday, 2/14, complete Tasks 1 and 2 of the project.
- By Monday, 2/17, complete Task 3 of the project, check your solutions against the rubric (included at the end of these instructions), and submit your files through Moodle.

This project is due on Wednesday, 2/19, by 11:59pm.



```
'string' * 2
```

expression vs literal

an expression does operations on literals

order of operations, what?

We'll see more examples of this during class today

are videos required?

Nope! Just an alternative format for information

code block reassignment was a bit tricky

As you see more examples, this will make more sense. It'll come up on HW2, so go to OH or TA if you want an in-depth explanation

COLAB WORKBOOK

Link: click for access

DISCUSSION
workbook

APPLE BAGGING

Diana is writing a program to help an apple orchard. She knows that the orchard gathered in total **42** apples this week, and then sold them in **bags of five**. She adds some calculations to her program that she thinks would be helpful.

```
42 / 5 = 8.4
```

Context: There are **exactly** **8.4 bags of apples in total**

```
42 // 5  = 8
```

Context: There are **8 full** **bags of apples**

```
42 % 5  = 2
```

Context: There are **2** **apples that are unbagged**

What is Diana calculating when she uses the **//** and **%** operators, and what is the context of that expression in terms of apples or bags of apples?

ARITHMETIC OPERATORS

when computing using operators, the order (precedence) matters (similar to PEMDAS in math)

( ) → ** → * / // % → + -

**

exponent

`2 ** 3 = 8`

*

multiplication

`7 * 4 = 28`

/

division

`7 / 4 = 1.75`

//

floor division (quotient)

`7 // 4 = 1`

%

modulus (remainder)

`5 % 3 = 2`

-

subtraction

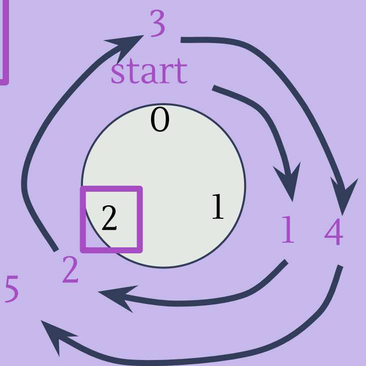
`7 - 4 = 3`

+

addition

`7 + 4 = 11`

helpful to visualize modulus as a clock



does not round up!

/

division

`7 / 4 = 1.75`

returns a decimal

//

floor division (quotient)

`7 // 4 = 1`

returns an integer

FLOATS

INTS

in Python, numbers are stored as ints (integers, whole numbers) and floats (decimals, floating point numbers) and these types are managed automatically

WHY BOTH? variety of ways numbers are used:
floats are good for calculations
ints are good for counting and iteration

follow along in workbook to see this in action

int() also does not round up!

Another way of thinking about this is that it is unaware of anything happening to the right of a decimal point

you can change from one to another in a process called **CASTING** using `int()` and `float()` functions which take another data type as input

```
float(5)     = 5.0

int(3.14)   = 3
int(3.9)    = 3

int(7/4)    = 1
```

1.75

```
float(7//4)= 1.0
```

1

DISCUSSION
workbook

PREDICT THE OUTPUT

1. `(1+2)**3` = (3)**3 = 3**3 = 27
2. `4 + 3 / 8` = 4 + 0.375 = 4.375
3. `int((4+3)/2)` = int(7/2) = int(3.5) = 3
4. `float(19//5)` = float(3) = 3.0
5. `3*(1//3)` = 3*(0) = 0
6. `1/3` = 0.333...333
7. `1.2 - 1` = ???

WHY? 1.2 has an infinite binary expansion, but computers have **finite memory**

this is not exactly 1/3 but rather an approximation

$$1/3 = 0.333\ldots333$$

WHY? decimal expansion for 1/3 doesn't end, but computers have **finite memory**

$$1.2 - 1 = 0.1999\ldots96$$

the exact value of 1.2 isn't stored in the computer

remember, computers think in binary! so we need to convert this decimal number to binary somehow

WHAT!!?!?!

CAUTION

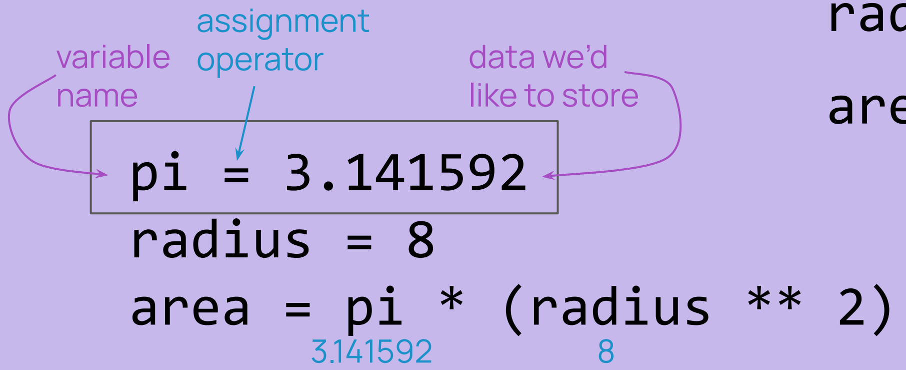
while we like to share ideas across disciplines, we need to be careful as **variable** and **=** mean different things than they do over in math world

~~3.141592 = pi~~

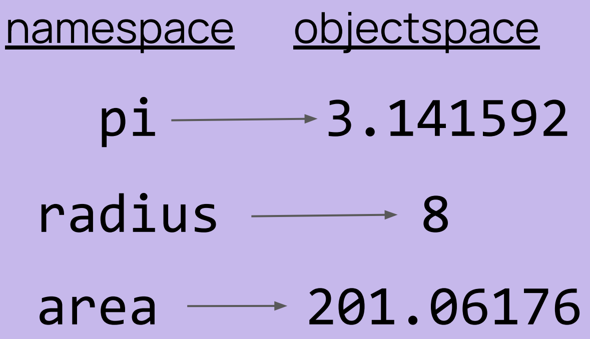
pi $\overset{\text{“gets”}}{\leftarrow}$ 3.141592

VARIABLES are used when we'd like to store a piece of **data** in memory, so that we can refer to it later on

numbers or strings



```
pi = 3.141592
radius = 8
area = pi * (radius ** 2)
```



Good variable naming convention is all lowercase with underscores if needed `sensible_variable_name`

PYTHON IS KINDA SMART

if you attempt to use a variable name that hasn't been used before Python will create it automatically (unlike some other languages)

`area` `Area` `aRea` `ar3a`

CAUTION: PICKY VARIABLE NAMES

Python is **case-sensitive** and will treat all of these variable names as separate entities. Variable names must **start with a letter or underscore**. You can include **numbers**, but **no other punctuation**

RESERVED WORDS should not be used for variable names

when these words come up, the interpreter that takes your Python code and converts to machine code already has a meaning attached to these!

and
as
assert
break
class
continue
def
del
elif
else
except
exec
finally
for
from
global
if
import
in
is
lambda
not
or
pass
print
raise
return
try
while
with
yield

```
“here’s a string”   ‘there’s a string’
```

FLOATS **STRINGS** **INTS**

data can be stored as numbers using the float and int types, or as text using the string type

OPERATIONS

spacing with concatenation is super important!

STRING CONCATENATION

```
“potato” + “potato” + “ !”  →  “potatopotato !”
    “potato” + str(3)  →  “potato3”
```

```
      “potato” * 2  →  “potatopotato”
```

```
“potato” - “tato”
   3 + “potato”      ERROR
```

you can change numbers to strings in a process called **CASTING** using **`str()`** function

```
str(1)       = “1”
str(1.5)     = “1.5”

          3
str(int(3.9)) = “3”
```

but strings cannot be cast as float or ints

```
int(“hi”)    ERROR

float(‘hi’)  ERROR
```

note the spaces! super important to
remember, especially for Standard IO!!

```
name = input(“What is your name? ”)
print(“Hello ”, name)
print(“Hello ” + name)
print(f“Hello {name}”)
```

the **print** function can take strings, floats, ints, and variables as arguments

the **input** function deals in strings: strings as arguments, strings as output

```
What is your name? Eowyn
Hello Eowyn
```

this is printing with what is called an **f-string**. this is something you do NOT need to know unless it’s of interest to you

```
number = int(input(“Enter a whole number: ”))
squared = number ** 2
print(“The square of”, str(number), “is”, squared)
print(f“The square of {number} is {squared}”)
```

```
Enter a whole number: 2
The square of 2 is 4
```

FUNCTIONS SO FAR

FUNCTION	ARGUMENTS	RETURN VALUE	NOTES
`int()`	float or string	int	converts argument to int
`float()`	int or string	float	converts argument to float
`str()`	int or float	string	converts argument to float
`print()`	text to be displayed	**None**	displays argument
`input()`	text prompt for user	string	

TIME FOR TODAY’S WORKBOOK

General workflow:

- Put all name cards face down.
- Start with concept checks, discussing first with partner and then with table group.
- When done, flip all name cards up and move onto exercises.

BEFORE NEXT TIME

Finish workbook

Good start on HW 1