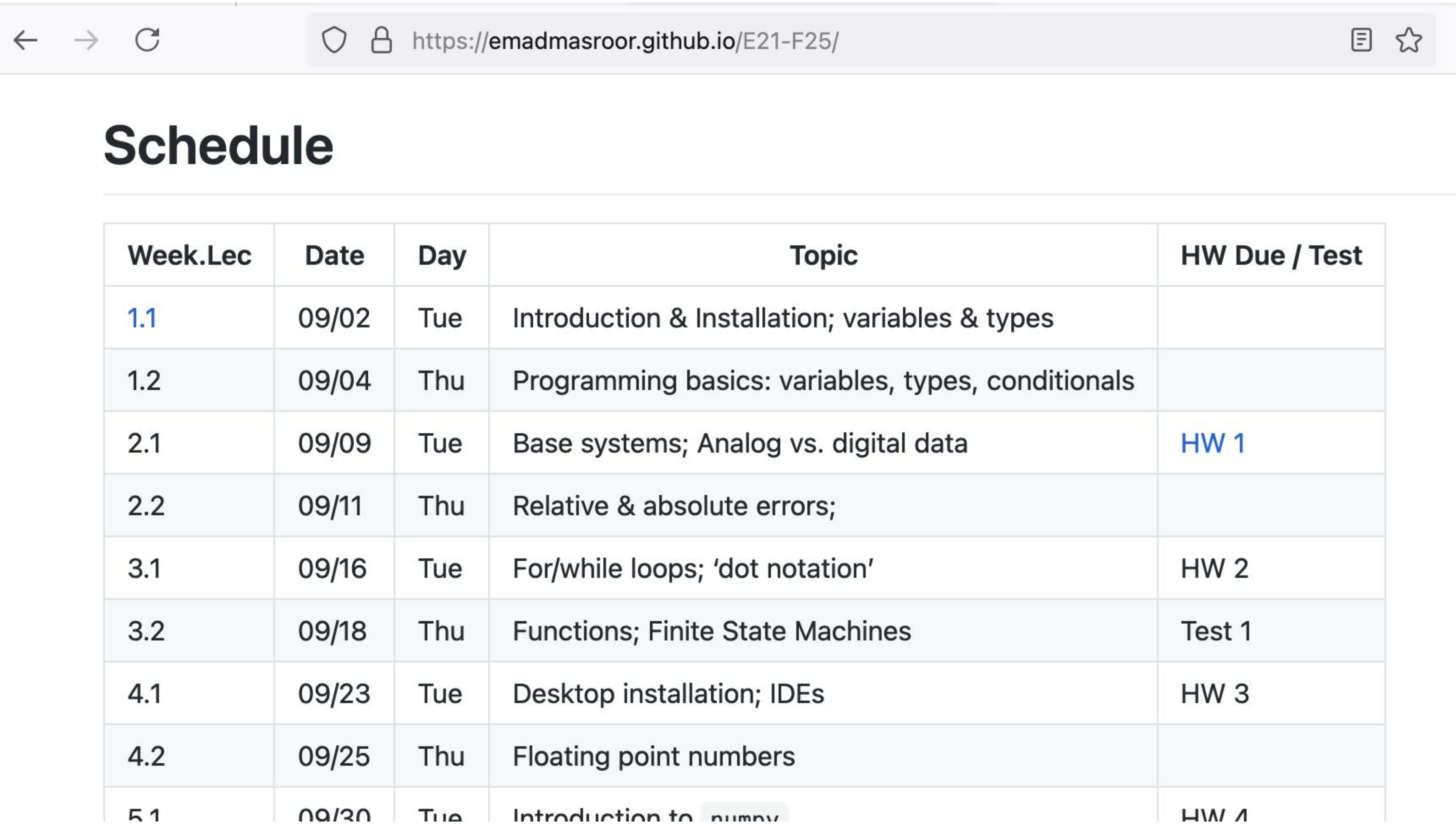
ENGR 21:
Computer Engineering Fundamentals

Lec 1.2
Thu Sep 4, 2025

Accessing Homework & Lectures


https://emadmasroor.github.io/E21-F25/

Schedule

Week.Lec	Date	Day	Topic	HW Due / Test
1.1	09/02	Tue	Introduction & Installation; variables & types	
1.2	09/04	Thu	Programming basics: variables, types, conditionals	
2.1	09/09	Tue	Base systems; Analog vs. digital data	HW 1
2.2	09/11	Thu	Relative & absolute errors;	
3.1	09/16	Tue	For/while loops; 'dot notation'	HW 2
3.2	09/18	Thu	Functions; Finite State Machines	Test 1
4.1	09/23	Tue	Desktop installation; IDEs	HW 3
4.2	09/25	Thu	Floating point numbers	
5.1	09/30	Tue	Introduction to `numpy`	HW 4

Variables and Types in Python

Python Preliminaries

- Python is an interpreted language.
- Typically, extra spaces don’t matter
- Indentation matters
- Case matters
- Comments start with ‘#’ or are enclosed within ‘’’triple inverted commas’’’
- Pressing enter in the REPL executes that line; use semicolons to indicate successive commands

```
>>> a=5
>>> a = 5
>>> a   = 5



>>> a =5
>>>  a=5
IndentationError: unexpected indent

>>> a = 5
>>> print(A)
NameError: name 'A' is not defined. Did
you mean: 'a'?

>>> p = 5; q = p + 4; print(q)
9

>>> a = 5 # there are 5 apples
```

Variables in Python

‘Assigning’ a variable `x = 5`

The main types of variables in Python:

1. Strings: `str`
2. Numbers: `int, float, complex`
3. Boolean: `bool`
4. Sequences: `list, tuple, range`

Some other types: `set, bytes, NoneType`

Check the type of a variable using `type(x)`

The `str` type

A string is a collection of characters.

Python provides several built-in functions to manipulate and interpret strings. For example:

- `upper` – returns a string with the letters capitalized
- `islower` – tells you whether the string is lowercase
- `find` – looks for a character inside a string

```
>>> x = “This is a string”
>>> x = ‘this is also a string’

>>> a = “hello!”

>>> a.upper()
“HELLO!”

>>> str.upper(a)
“HELLO!”

>>> str.find(a,”o”)
4

>>> a.find(“o”)
4
```

Classes and methods

" Type " functions

We will use

- ➔ ‘class’ and ‘type’ interchangeably
- ➔ ‘method’ and ‘function’ interchangeably

type

In Python, each class is associated with certain methods.

functions

There are two ways of using these methods.

```
>>> X = “sample string”

>>> type(X)
<class 'str'>

>>> X.isnumeric()
False

>>> str.isnumeric(X)
False
```

X is an instance of class `str`

`isnumeric` is a method associated with class `str`

'isnumeric' is a built-in function for type 'str'

<type>.<function>

floating - point integer.

The `int` type vs the `float` type (& `complex` type)

- `int` is a signed integer
- `float` is the closest representation of a real number that a computer can contain.
- It's important to know whether a variable is an int or a float.
- You can convert between these types using `float(x), int(x), complex(x)`

```
>>> a = 5        # this is an integer
>>> a = 5.0      # this is a float

>>> complex(a)
5.0+0j

>>> a = 5+4j
>>> type(a)
<class ‘complex’>
```

The `bool` type

Boolean truth-values

Conduct logical operations

`and , or , not`

(4==3) is a boolean variable with value 'False'

Notice lack of quotes!

```
>>> a = True
>>> type(a)
<class ‘bool’>

>>> 4 == 3
False

>>> 4 == 4
True

>>> 4 > 3.2
True

>>> c = 4==4
>>> print(c)
True
```

equals, NOT "assign"

The `list` type

from adafruit_circuitplayground import cp

An indexed collection of elements

The elements of a list can be any type – even other lists!

For the Circuit Playground Bluefruit, `cp.pixels` is (kind of) a list.

What are its elements?

Accessing the elements of a list by index

```
>>> a = [1,3,5,7,10]
>>> type(a)
<class ‘list’>

>>> a[0]
1
>>> a[3]
7

>>> b = [4, 2.4, 2+4j]
>>> type(b[0])
<class ‘int’>

>>> type(b[1])
<class ‘float’>

>>> type(b[2])
<class ‘complex’>
```

The `tuple` type

An indexed collection of elements that is **immutable**.

The elements of a tuple can be any type – even other tuple!

Often used to provide multiple arguments.

```
>>> a = (1,3,5,7)
>>> type(a)
<class ‘tuple’>

>>> a[0]
1
>>> a[3]
7

>>> b = (4, 2.4, 2+4j)
>>> type(b[0])
<class ‘int’>

>>> type(b[1])
<class ‘float’>

>>> type(b[2])
<class ‘complex’>
```

The `range` type

An abstract type that refers to the range of numbers (integers) between two numbers.

Take care with Python's indexing!

```
>>> a = range(3)
>>> a
range(0, 3)
>>> a[0]
0
>>> a[1]
1
>>> a[2]
2
>>> a[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: range object index out of
range

>>> range(5) == range(0,5)
True  in Circuit Python, false

>>> range(3.3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be
interpreted as an integer
>>>
```

Conditionals and `if` statements

Probably the most important thing a computer can do

Conditionals check whether something is true

A piece of code that evaluates to one of the two Boolean values,

`True` or `False`

```
>>> 4 == 4
True

>>> 4 == 3
False

>>> 4 != 3
True

>>> 4 != 4
False

>>> 3.9 < 4
True

>>> 4.1 >= 4
True
```

Some edge cases for conditionals

- Use Circuit Python in ‘interactive’/REPL mode to determine how Python behaves in these scenarios.

```
>>> True == ‘True’          False
>>> "abc" < "def"           True
>>> “abc” < “ab”            False
>>> “abc” < “aba”           False
>>> “A” < “a”               True
>>> [1,2,3] < [2,3,4]
>>> [1,2,3] < [2,3,2]
>>> (1, 2, 3) == [1, 2, 3]  False
```

```
>>> 3 == ‘3’                False
>>> None == 0               False
>>> None == None            True
>>> 0.1 + 0.2 == 0.3        True
>>> 1 == 0.9999             False
>>> 1 == 0.999999...  with enough 9's
>>> (3 + 2j) > (2 + 1j) → error
>>> [] == []
>>> ‘’ == ‘’
>>> [] == ‘’
>>> [1,2] < [3,4,5]
>>> True == 1               True
>>> False == 0              True
```

The Python `if` statement

Remember indentation!

What goes in here must evaluate to True or False

```
if (conditional):
    execute some code only if condition is true
```

Need ":"

Need indent

Notice no "end", as in MATLAB

Q. How big does the indent need to be?

Always use monospaced fonts

```
x = 5
if x < 6:
     print(“yes”)
```

When working with files

```
>>> x = 5
>>> if x < 6:
...     print(“yes”)
...
```

When working in the REPL

What comes after `if`?

- “Else”
- “Else, if”
- No limit to number of ‘elifs’
- Only one ‘else’.

```
if x < 5:
     print(“x is less than 5”)
else:
     print(“x is not less than 5”)
```

same indentation

```
if x < 5:
     print(“x is less than 5”)
elif x > 6:
     print(“x is more than 6”)
```

```
if x < 5:
     print(“x is less than 5”)
elif x > 3:
     print(“x is more than 3”)
else:
     print(“x is neither < 5 nor > 3”)
```

Nested if’s

There is no restriction on placing `if` statements inside other `if` statements.

Logical operators in Python

There are 3:

- `and`
- `or`
- `not`

Use parentheses to group together longer logical operations

```
>>> 2 < 3 and 3 < 4
True

>>> 2 < 3 or 10 < 9
True

>>> not True
False
>>> not False
True
```

Try running some of these in the REPL

```
>>> 2 < 3 or 'a' < 'b' and 'a' < 'A'

>>> 2 < 3 or ('a' < 'b' and 'a' < 'A')

>>> (2 < 3 or 'a' < 'b') and 'a' < 'A'
```