Lecture 04

Program Input and Number Systems

14 Pluviôse, Year CCXXX

Song of the day*: *Rally, Rally (fear. Pii & meiyo) *by MAISONdes (2021).*

In Monday's class we learned that we can display the values of variables and expressions by means of the `print()` function:

```
lecture_id = 8
print(lecture_id)

message = "オマエはもう死んでいる。"
print(message)

obvious_fact = 5 != "5"
print(obvious_fact)
```

Output:

```
8
オマエはもう死んでいる。
True
```

That's a great thing to be able to do, and we'll be making ample use of this faculty. However, what kind of programs would we realistically be writing if we weren't able to interact with our user? After all, almost every program that is useful to us in some way gets our input; your phone registers your touch as an input, your laptop registers every key stroke as an input, a camera registers light as input. Input, input, input.

It stands to reason, then, that this should be the next thing we need to focus on.

Part 1: *Program Input*

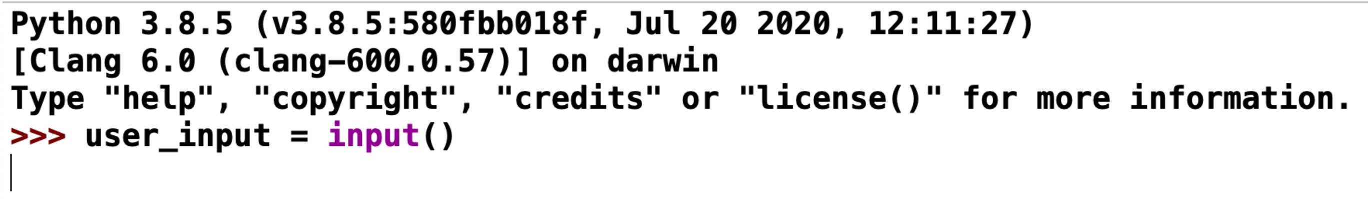
The most basic form of user interaction in Python is done through a very succinctly named built-in function— `input()` .

At its most basic level, it functions as follows:

```
user_input = input()

print(user_input)
```

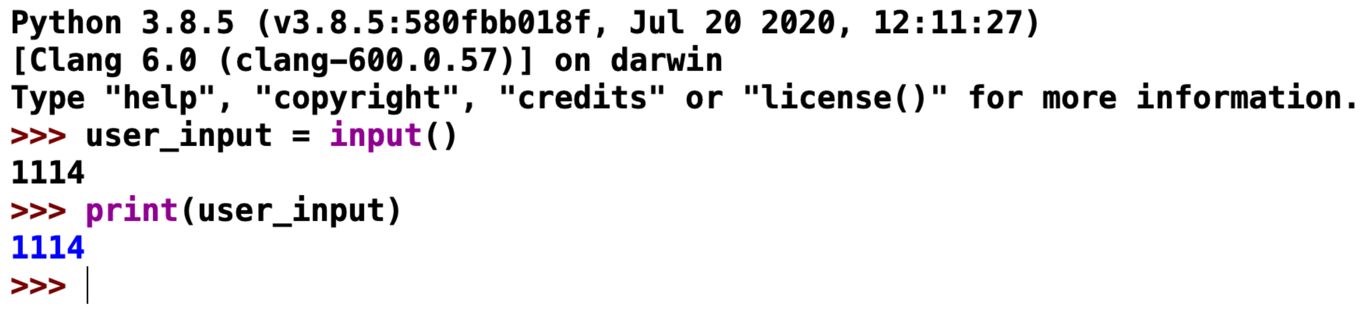
If we run this program, you will see that our shell window will pause, and wait for an action from us:



```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> user_input = input()
```

***Figure 1**: Our shell prompting us for input.*

If we type something in—say, the course number for this class—and press "enter", you will see the following behavior:



```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> user_input = input()
1114
>>> print(user_input)
1114
>>> 
```

***Figure 2**: Our shell displaying our input.*

This works just fine. But typically speaking, we want our programs to be as intuitive and user-friendly as possible—to have good **UI** and **UX**, in other words. The `input()` function allows us to give the user a "prompt" message by putting it, ***in string form***, inside the `input()` function's parentheses:

```
course_number = input("What is this class's course number? ")

print(course_number)
```

If we ran this, our shell would prompt us the following way:

```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> user_input = input("What is this class's course number? ")
What is this class's course number? 
```

__Figure 3__: Our shell prompting us for this class's course number.

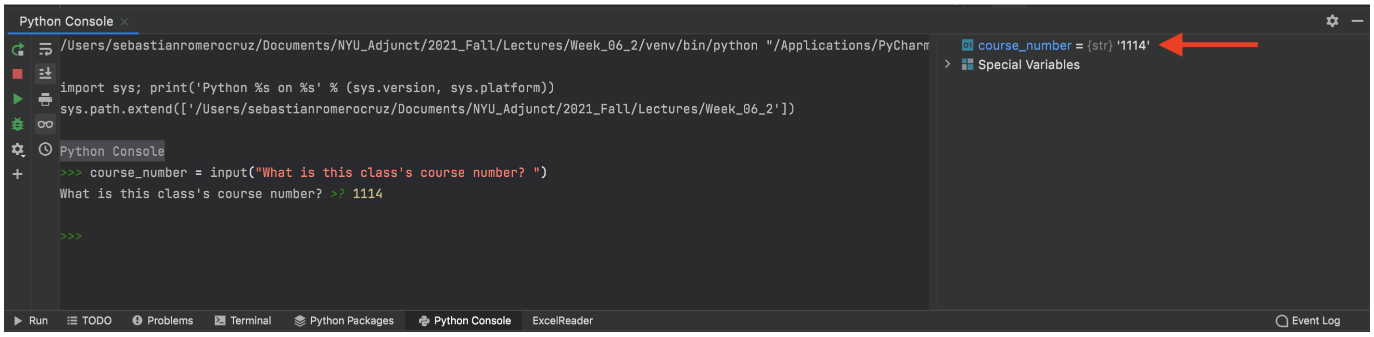
Once we enter our desired input and press the "enter" key, we will see the following:

```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> user_input = input("What is this class's course number? ")
What is this class's course number? 1114
>>> print(user_input)
1114
>>> 
```

__Figure 4__: Our shell displaying this class's course number.

These two programs, effectively, do the same exact thing (i.e. accepting user input and displaying), but in the first one, we are barely even aware that we're being prompted for input—and we have no idea what input is supposed to even *be*. The second example, by contrast, at the very least gives us a clear idea of the type and nature of our input. It won't stop any user from entering the wrong thing, but at least we can say that we gave them some hints.

Now, interestingly, **Python saves all input in `str` form**, meaning that our input of "1114" is not saved as an integer, as one might expect, but as a string. Sure enough, if we run the same code on our console, we can very clearly see that the variable `course_number` is a `str` object:



```
/Users/sebastianromerocruz/Documents/NYU_Adjunct/2021_Fall/Lectures/Week_06_2/venv/bin/python "/Applications/PyCharm

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/sebastianromerocruz/Documents/NYU_Adjunct/2021_Fall/Lectures/Week_06_2'])

Python Console
>>> course_number = input("What is this class's course number? ")
What is this class's course number? >? 1114

>>>
```

```
course_number = {str} '1114'
```

__Figure 5__: PyCharm's console displaying the type of `course_number` on the right.

There is essentially no way of changing this behavior. Python, by design, received all input in string form. It's up to us, the programmers, to parse that input into a usable form.

Part 2: *Number Systems*

What do we mean when we say "magnitudes bigger than", or "magnitudes smaller than"? Mathematically speaking, for something to be a magnitude larger or smaller than another thing, they have to follow the same number system (i.e. you can't compare apples to oranges, etc.)

Since the world today primarily uses the **decimal system** for mathematical calculations, when we say `a` is (for example) "3 magnitudes" larger" than `b`, this is what we mean:

> $a = b * 10^3$

So, if `b` equal **42**, `a` would equal **42,000**.

In other words, the number we use to differentiate magnitudes in the decimal, or **base-10**, system is the number **10**. We can also tell that this is the case because **there is no single digit to represent 10**. We, instead, have to write out ten as a combination of two digits, 1 and 0.

The same is the case for 100. Since we don't have a symbol for 10, we cannot represent 10 tens in a two digit format. Therefore, we need to use three digits to go up a magnitude (1, 0, and 0).

Another way of thinking of base-10 numbers is as the **sum of numbers multiplied by the powers of 10**:

> $4{,}034 = 4 * \mathbf{10^3} + 0 * \mathbf{10^2} + 3 * \mathbf{10^1} + 4 * \mathbf{10^0} = 4000 + 0 + 30 + 4$

But what about other number systems? Computers, for instance, do use the decimal system to count or do mathematical operations. The reason for this is that computers can only do operations in **ones and zeros**. This number system uses **2** instead of 10 to differentiate between magnitudes. We would, thus, call this system ***binary***, or ***base-2***.

To count from 1 to 10 in binary, then, we would do the following:

Decimal	Binary
0	0
1	1
2	10

Decimal	Binary
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

***Figure 1**: Counting to 10 in **binary**.*

Just like decimal assumes that we don't have a symbol for 10, **binary assumes that we don't have a symbol for 2**. Therefore, we must count using only the numbers under **2** (i.e. 0 and 1).

Again, just like decimal can be represented as a sum of numbers being multiplied by the powers of 10, **binary numbers can be represented as a sum of number being multiplied by the powers of 2**. For example:

> $(1001)_2 = 1 * \mathbf{2^3} + 0 * \mathbf{2^2} + 0 * \mathbf{2^1} + 1 * \mathbf{2^0} = 8 + 0 + 0 + 1 = (9)_{10}$

We can better illustrate this with the following **web tool**:

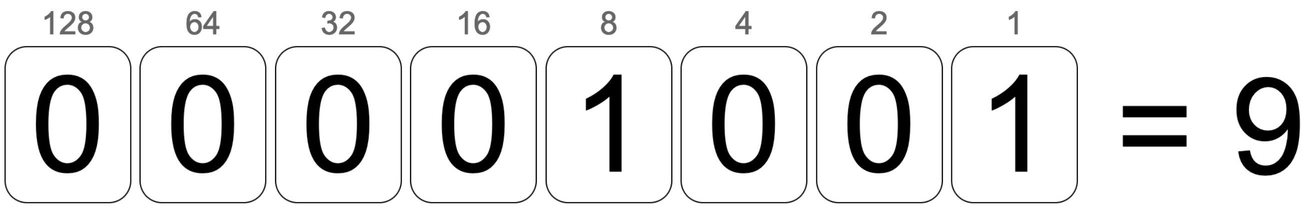


***Figure 2**: A visual representation of $(1001)_2$ to $(9)_{10}$.*

If you're given a number in decimal, and have to convert to binary, you can do a simple division and take note of the remainders, which end up being your binary number. For example, if we wanted to convert 29 to binary:

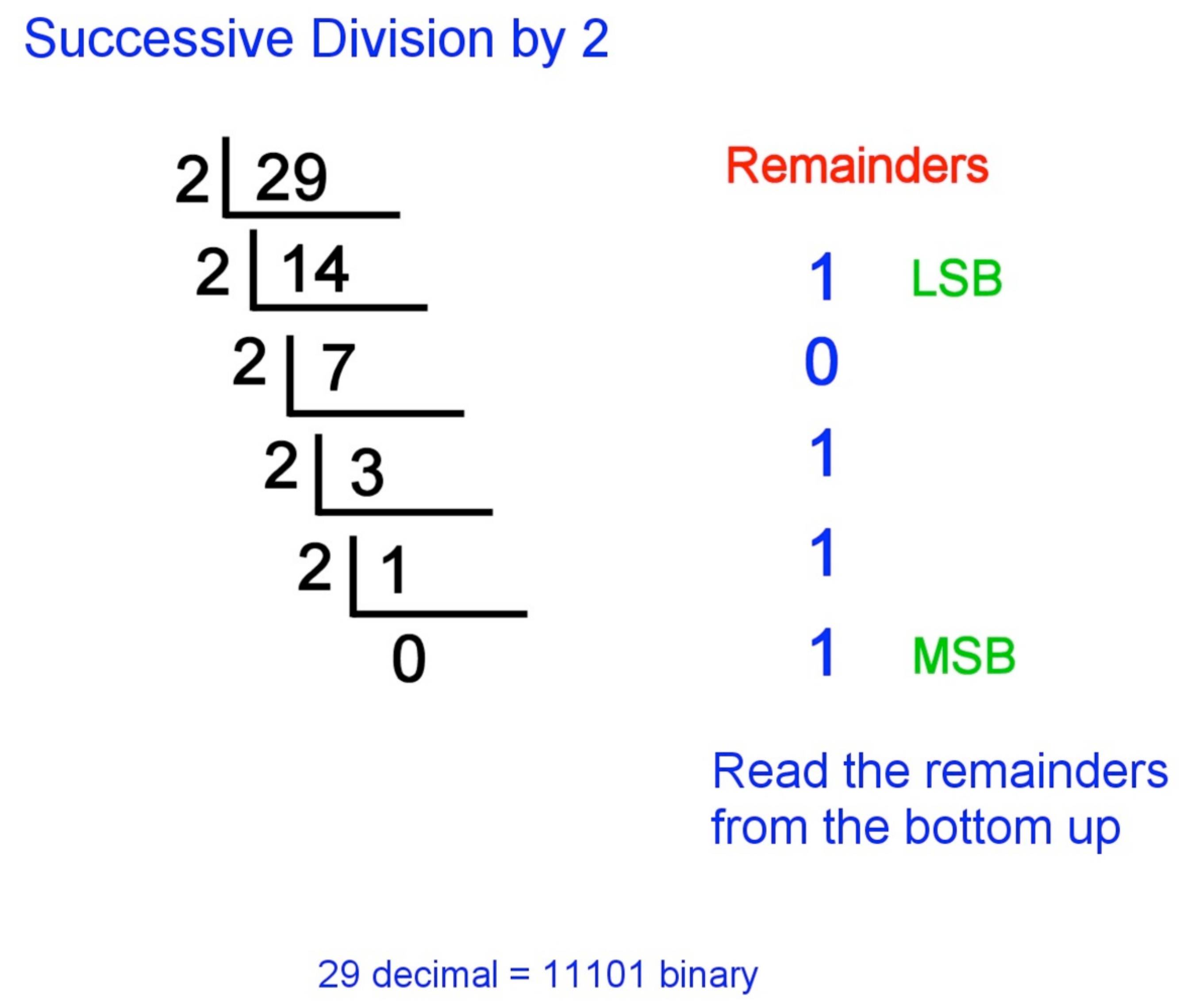


Figure 3: *Converting $(29)_{10}$ to $(11101)_2$ (**source**).*

You can do this process for just about any number system. Let's try a base-5 system for funsies.

Decimal	Base-5
0	0
1	1
2	2
3	3

Decimal	Base-5
4	4
5	10
6	11
7	12
8	13
9	14
10	20
11	21
12	22
13	23
14	24
15	30
16	31
17	32
18	33
19	34
20	40
21	41
22	42
23	43
24	44
25	100
26	101
27	102

Decimal	Base-5
28	103
29	104
30	110

Figure 4*: Counting to 30 in base-5.*

Not all number systems are equally relevant to computer science, however. The second most important base to be aware of is base-16, or **hexadecimal**.

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	a
11	b
12	c
13	d
14	e
15	f

Decimal	Hexadecimal
16	10
17	11
18	12
19	13
20	14

***Figure 5**: Counting to 20 in **hexadecimal**.*

As you can see, because in Arabic numerals we don't have a symbol for anything larger than 9, and because we're not allowed to use 10 until go up in magnitude, we have to resort to using tokens for these values. You can literally use whatever symbol you want as a token, but the well-established convention is to use the letters `a` through `f`.

Why is hex relevant to computer science?

Memory is stored inside your machine in units called **bytes**, which are themselves usually composed 8 bits of **bits** (a 1 or a 0). Let's say we have one such byte below:

```
10111001
```

The reason why hexadecimal is helpful here is that, if we split this byte into two units of 4 bits:

```
1011 1001
```

Each of those 2 halves can give us a **total of 15 possible bit combinations** before growing in size (magnitude):

1. `0000`
2. `0001`
3. `0010`
4. `0011`
5. `0100`
6. `0101`
7. `0111`
8. `1000`
9. `1001`

10. `1010`
11. `1011`
12. `1100`
13. `1101`
14. `1110`
15. `1111`

What system can only count to 15 before growing in magnitude? Hexadecimal! This, consequently, makes conversions between decimal and hexadecimal extremely simple *if you know your binary equivalents*:

Let's say we wanted to convert $(1967)_{10}$ to both binary and hex.

Using repeated division by 2, we get:

> 1967 / 2 = 983.5 —> **1**
>
> 983 / 2 = 491.5 —> **1**
>
> 491 / 2 = 245.5 —> **1**
>
> 245 / 2 = 122.5 —> **1**
>
> 122 / 2 = 61 —> **0**
>
> 61 / 2 = 30.5 —> **1**
>
> 30 / 2 = 15 —> **0**
>
> 15 / 2 = 7.5 —> **1**
>
> 3 / 2 = 1.5 —> **1**
>
> 1 / 2 = 0.5 —> **1**

Reading from the bottom up, we get **$(11110101111)_2$**. Now, split this binary number into groups of four

> **0111 1010 1111**

The decimal equivalents of these three groupings are easy to figure out: **7**, **10**, and **15**. By the same token, finding the hexadecimal equivalents of 7, 10 and 15 isn't difficult if we are familiar with counting in hex (figure 4):

> $(1967)_{10} = (11110101111)_2 = \mathbf{(7af)_{16}}$.

And just like that, we converted 1967 to all relevant units in one fell swoop!