Lecture 0:

Monday, August 12, 2024 7:30 PM

Functions + Variables

$ code hello.py
- begins a python program

- print("hello, world")
  - run with "python hello.py"
  - ↳ Python is an interpreter that handles the process of reading + running.

What just happened?
- you passed an argument to the print() function, and the program ultimately outputs to cmd.

What if you forget the closing parenthesis?
- big mistake, SyntaxError.
  Open parenthesis was never closed, you need to type very literally in programming.

Improving Hello World

Ask, Input, Present → print() input() print()...

What's your name? — [name] — Hello, [name]

improvement...
- "input()" can have a prompt arg.
- functions can have return values. Input() can return the value that was inputted to it by the user.
- use a variable!

```
name = input("What's your name?")
```

variable ← name, assigns ← =, function ← input
container "name" is variable that stores input from function.

proper syntax

```
name = input("What's your name?")
print("hello,")
print(name)
```

"#" is used to add comments
e.g. #ask user for their name

Comments can serve as reminder texts, to-do lists, even pseudocoding.

""" comment area """

... intput() expects only a string

Further improving..

```
name = input("What's your name?"
print("hello,")
print(name)
```

into... `print("hello, ", name)`

note: when you pass multiple arguments to print, a space is automatically added.

input: "Leo"
output: "hello, Leo"

On print()...

print() automatically adds a new line at the end
By reading documentation, we find out we can pass certain arguments into its parameters to prevent it.

*objects means print() can take any number of strings.
sep=' ' means arguments passed are separated by a space.
end='\n' means by default every print() ends in newline.

Lets override these...

```
name = input("What's your name?")
print("hello, ", end="")
print(name)
```

output = "hello, Leo"

most elegant solution...

use an fstring ← Alerts python it's a special string.

```
print(f"hello, {name}")
```

input validation on this string.
Luckily, strings come with really cool functionallity
What will happen if user types "   Leo"
↳ weird "   Leo" output
Let's remove these spaces...

This removes all whitespace from the name variable

```
name = name.strip()
```

Now, "   Leo" input prints "Leo".
Let's force capitalization...

This capitalizes first character of string name

```
name = name.capitalize
```

input = "leo" output = "Leo"
but... input = "leo buj" is output = "Leo buj"

simply, `name = name.title()`

Now chain it together...

```
name = name.strip().title
```

We can even do...

```
name = input("What's name?").strip().title()
```

Stylization is important

Consider the readibility of your code.
Learn patterns and styles that allow not only yourself, but colleagues to read your code.

Let's separate the user's first & Last name

```
first, last = name.split(" ")
```

(neat thing is python interactive mode)

Transitioning To Integers

\+ − * / %

calculator.py

```
x = input("What's x? ")
y = input("What's y? ")
z = x + y
print(z)
```

We have an issue here... inputs x + y are strings that are being concatenaded.

e.g. x = "1" y = "2"
z = "1" + "2" = "12"

Fixing this... (converts to an integer!)

```
z = int(x) + int(y)
```

We could also nest these functions...

```
x = input("......")
x = int(x)
```

→ `x = int(input("What's x? ")`

This is an improvement, and showcases how we may be more efficient by nesting our method calls.
It can be bad though...
by overnesting, we can make too many nested calls and overcomplicating code.

float

decimal numbers!
Let's add support.

```
x = float("... x? ")   y = float("...y? ")
```

Now, let's round the result...

```
round = (number[, ndigits])
```

* generally, this kind of notation implies an optional input

...

```
z = round(x+y)
```

input x = 1.2 + y = 3.4 would output 5.

What about formatting our output?
e.g. instead of "1000", output "1,000".

```
print(f"{z}")
```

evolve...

```
print(f"{z:,}")
```

This accomplishes this formatting goal.

floats WILL round eventually.

Let's pivot to dividing x by y
x/y, 2/3 outputs to 0.6666666666
How about we round?

```
z = round(x/y, 2)
```

This outputs to 0.67

Another way!

```
print(f"z:.2f")
```

This is how you specify with an fstring how many decimals to round to.

Focusing on FUNCTIONS

```
name = input("What's your name? ")
hello()
print(name)
```

This function doesn't exist! (yet)

Let's make it a reality.

```
def hello():
....print("hello")
```

Now the function exists, and its correct!

We can customize our function with parameters

```
def hello(to):
.... print("hello ", to)
```

We're able to plug into our function call the name input by simply hello(name)
nuance.. the variable name is copied to to

Add a default value just in case the hello() is called with no argument.

```
def hello(to="world")
.... print("hello, " to)
```

hello() would print "hello, world"

function definitions must be defined before the main part of your code (where they're called).
Understand that variables only exist in the context that it was declared in.

```
main:            def leo():
  x = 2            y = 2
```

* y only exists inside of main.

You can also have a function return a value with return keyword.