

# Introduction to Python

By: Abhay Sheshadri, Daniel Alex, Vismay Igur, Suyog Khanal



Day 1

# Agenda

- What is Python?
- Repl.it
- Commenting
- The print() function
- Operators
- Variables
- Data types

# What is Python?

- It is a high level and general programming language with many purposes. You can easily use other people's code in your own projects:
  - Can be used to make games (pygame, Coco3D)
  - Can be used to make AI (tensorflow, pytorch, Sci-kit learn)
  - Can be used to do web scraping (BeautifulSoup4)
  - You can also make a discord bot (discord.py)
- Easy to understand compared to other languages. Looks more like english than assembly.

# Why Should You Learn Python?

- It's very simple to use.
- Its syntax is similar to other programming languages, making it easier to learn other ones in the future.
  - With knowledge of Python, you can learn languages like Java and C++ easily.
  - It can help you when you take computer science classes in high school.
  - Holds a bright future in the fields of machine learning and artificial intelligence.

# What You Will Learn

- Throughout this course, we will discuss the many features of the Python language including:
  - How to construct and implement:
    - Variables
    - Operators
    - Lists
    - Conditional statements
    - Loops
- In the end, we will be using these features to create a game of Tic Tac Toe!
- <https://repl.it/@HealableHades1/TicTacToe-Goal>

# Sign up for Repl.it

1. Go to [repl.it](https://repl.it)
2. Click “Sign up”.
3. Make an account by signing up with your school or personal email.
4. Click “Skip” under “let’s get started”.
5. Select “Python” for languages you are interested in.
6. Under “or start coding” click “Python”.

Please link your repl.it by pressing “Invite” at the top of the screen and copying your link when you have a question.

Feel free to ask questions throughout in the chat if you need help!

# Printing Your First Program

- Used to display text onto the screen
- Use `print("any text you want")`
- Can also print numbers and other data types too, but you cannot use quotes for these.

Example:

```
tuts.py
1  print("hello world")
2  print("Hey, how are you")
3  print("What's going on?")
```

- You just wrote your first program! Hit run in order to execute your code.
- Your console should look like this:

```
hello world
Hey, how are you
What's going on?
```



# Commenting

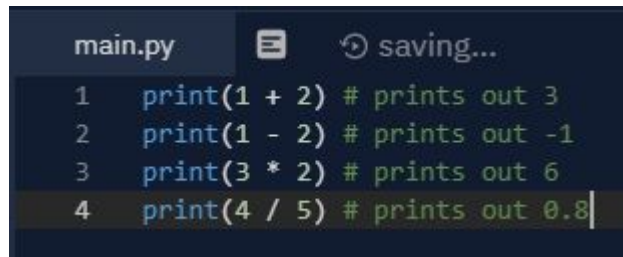
- Add notes to your code that does not run
  - Can be used to remind you what specific parts of the program do
- `#` for one lined comments
- `"""` for multiline comments (beginning and end)

Example:

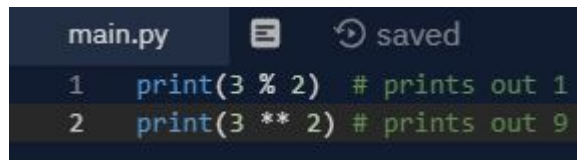
```
tuts.py
1  # This is a one line comment
2
3  """
4  This is a
5  multiline comment|
6  """
```

# Operators

- + - \* / do what you think they do.
  - + is addition
  - - is subtraction
  - \* is multiplication
  - / is division
- Other operations:
  - % is modulo. It outputs the remainder that you would get if you divided the numbers.
  - \*\* is for exponents.
- Order of Operations (PEMDAS)
- \*\* Challenge for homework \*\*
  - There are two ways to get square roots in python. Try to find both ways.
  - \*\*\*\* One of the ways might require algebra



```
main.py  saving...
1  print(1 + 2) # prints out 3
2  print(1 - 2) # prints out -1
3  print(3 * 2) # prints out 6
4  print(4 / 5) # prints out 0.8
```



```
main.py  saved
1  print(3 % 2) # prints out 1
2  print(3 ** 2) # prints out 9
```

# Variables

Think about it like this:

- A variable is a “box” that can store a data value.
- These values can be words, numbers, true or false values, and even lists that have more values.
- The left side is where you name the variable. (Can be any combination of letters and numbers)
  - A variable name cannot start with a number and cannot have symbols other than underscores.
- The right side is your new value. (Must be of one of the data types)
  - Do not need to specify data type.
  - Example: 

```
1 test = "dog"
```



# Variables (cont.)

- You can also update variables
- By setting the variable equal to a new value, you are taking the previous value out of the “box” and putting the new one in.
- By putting a variable in print(), you are printing out the value inside the variable at the time of execution.

```
main.py  [icon]  saved
1  # Setting the variable to a value
2  my_variable = "hello, how are you?"
3  print(my_variable)
4
5  # Updating the variable
6  my_variable = "I am good."
7  print(my_variable)
```

```
hello, how are you?
I am good.
> [ ]
```

# Data Types

- Data Types represent a value, whether they are words, numbers, whether something is true or false, or even a list of values.
- You can perform different types of operations on different data types.
- Some operations are exclusive to certain data types while others can be used with a variety of different data types.
- The `print()` function can be used with any data type, and all data types can be assigned to variables.

# Strings

- Strings are a data type where we can store text
  - `***` Text can also include numbers `***`
- They must always be inside a pair of quotation marks.
  - You can use both single (') and double quotes (").
- They can be set to a variable, like other data types.

```
main.py  [icon] saved
1  my_string = "Hello world"
2  my_second_string = 'hey world'|
```

# Strings (cont.)

- Strings can even be added to one another and multiplied by an integer, like numbers!
  - This is called concatenation.

Example:

main.py



saved

```
1 String1 = "Hello! My name is "  
2 String2 = "Mark. "  
3 print(String1 + String2)  
4 print(String2 * 4)  
5
```

```
Hello! My name is Mark.  
Mark. Mark. Mark. Mark.
```



# Integers/Floats

- Data Type that contains a number.
- Integers are whole numbers and must stay as whole numbers
- Floats are decimals.
- Performing an operation between a float and an integer results in a float.
- Example:

```
main.py  saved
1  wholeNumber = 3
2  decimal = 5.0
3  print(wholeNumber + decimal)
```

Outputs:

```
8.0
❏
```

Notice how it printed a float since the float has higher precision



# Booleans

- A Data Type that only has two possible values: True or False
- Used to represent a condition
- Example:

```
main.py  saving...  
1 trueOrFalse = False  
2  
3 print(trueOrFalse)
```

Outputs:

```
False  
❏
```

# Lists

- A list is mutable collection of objects that are in a sequence.
  - Objects in lists are known as items.
- These items can be of any data type: integers, strings, booleans, and even other lists.
- Like strings need to be inside quotation marks, all items in a list must be in square brackets.

main.py



⌂ saving...

```
1 list1 = ["Matthew", 20, True, 23, "Money", True]
2 print(list1)
3
4
```

```
['Matthew', 20, True, 23, 'Money', True]
>
```

# Indexing

- Indexing stores letters/numbers in a sequence.
  - But it is a little different from the traditional way of counting.
- In indexing, the first letter or item is always 0.
- Each box in the array is treated as a variable.
- Indexes are implemented by using square brackets. See example on the next slide.

# Indexing (cont.)

Example:

```
main.py  saving...
1  phrase = "Cats are not mammals."
2  friends = ["Jerry", "Maria", "Mark", "Dave", "Suzy"]
3
```

- phrase[0] would be "C" (from Cats^).
  - Likewise, phrase[10] would be "o" (from not^).
- friends[0] would be "Jerry".
  - And friends[3] would be "Dave".

# Indexing (cont.)

- In addition to finding an item or letter from a specified index, we can also do the reverse.
  - That is, finding the index of an item, letter or even string.

Example:

```
main.py  saved
1 phrase = "Cats are not mammals."
2 friends = ["Jerry", "Maria", "Mark", "Dave", "Suzy"]
3 print(friends.index("Maria"))
4
```

```
1
2 []
```

- In the example, we are trying to find the index of the item “Maria” in the list “friends”, should it exist.
- By using the .index() function, we can find the indices of specific items in a list.

# The len() function

- The len() function can be used to find the amount of items in a list.
- It can also be used to find the number of characters in a string.
  - This count includes spaces and punctuation.

Example:

```
main.py  saved
1 phrase = "Cats are not mammals."
2 friends = ["Jerry", "Maria", "Mark", "Dave", "Suzy"]
3 print(len(phrase))
4 print(len(friends))
5
```

```
21
5
[]
```

- By using the len() function with the string "phrase" and list "friends", we found out that there are 21 characters in the string and 5 items in the list.

# Exit Ticket

Complete this by next class (Also in chat):

<https://tinyurl.com/MonroePythonDay1>

Which of these work to square root the variable x?

- ☐  $x * 0.5$
- ☐  $x / 0.5$
- ☐  $x ** 2$
- ☐  $x ** 0.5$
- ☐ `math.sqrt(x)` (After importing math)
- ☐  $x - 0.5x$

Day 2



# Daily Agenda

- Review of Day 1
- The input() function
- Data type casting functions
- Logical operators

# Input

- In order to get input from the console in python, you must use the `input()` function.
- The `input` function gives a message prompting the user to enter some alphanumeric text. The program will be stopped when the `input()` function is running.
- Once the user enters the text and hits enter, the program will resume and the function will give you a string containing the user's text.



```
main.py  [icon] saved
1  name = input("Enter your name: ")
2
3  print("Hello " + name + "!")
```

# Casting

What happens if we try to combine a string with a number or any other data type like in the example below?

```
main.py  [icon]  ↻ saving...  
1  wholeNumber = 3  
2  print("There are " + wholeNumber + " apples")  
3
```

We get this error:

```
Traceback (most recent call last):  
  File "main.py", line 2, in <module>  
    print("There are " + wholeNumber + " apples")  
TypeError: can only concatenate str (not "int") to str  
✚ □
```

# Casting (cont.)

- So how do we combine other data types with strings?
  - **We must cast the data type**
- Use `str(other data type)` to convert a data type into a string

```
main.py  [icon] saved
1  wholeNumber = 3
2  print("There are " + str(wholeNumber) + " apples")
3
```

Now it works!

```
There are 3 apples
❖ □
```

Strings **must** combine with other string, so we convert other data types into string

# Casting with Inputs

Let's see what happens if we perform an operation on an input like below:

```
main.py  [icon] saved
1  wholeNumber = input("Enter a whole number")
2
3  print(3 + wholeNumber)
4
```

We entered 3, but when we added the inputted 3 in the print statement, we get this error:

```
Enter a whole number 3
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print(3 + wholeNumber)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
❗
```

# Casting with Inputs (cont.)

- This is because anything that is inputted becomes a string
- Like how we cannot combine strings with numbers, we cannot perform mathematical operations with strings
  - Again, we can solve this problem with casting!
- Use `int(String representing a whole number)`
  - Turns the string into the number (assuming the string represents the number, such as "7")
- When we cast, it works:

```
main.py  [icon] saved
1  wholeNumber = input("Enter a whole number")
2
3  print(3 + int(wholeNumber))
4
```

```
Enter a whole number 3
6
[icon] [ ]
```

# For Loops

- For loops are used to iterate over a sequence a specific number of times.
- The structure of a for loop is as follows:

```
main.py  [icon]  ↻ saving...  
1  for x in list:  
2      # do something  
3
```

# For Loop Example

main.py



⌂ saving...

```
1 friends = ["Dave", "Mike", "Grace", "John"]
2 for friend in friends:
3     print(friend + " is awesome!")
4
```

```
Dave is awesome!
Mike is awesome!
Grace is awesome!
John is awesome!
❏
```

- Indenting makes all the difference in for loops.

main.py



⌂ saving...

```
1 friends = ["Dave", "Mike", "Grace", "John"]
2 for game in friends:
3     print(game + " is awesome!")
4
```

```
Dave is awesome!
Mike is awesome!
Grace is awesome!
John is awesome!
❏
```

- Notice how the name of the variable in the loop doesn't matter.
  - We changed the variable from "friend" to "game", but as long as we replace "friend" to "game" everywhere in our loop, the output should remain unchanged.



# The range() function

- The range() function allows us to count iterations in a for loop.
- It also makes for a simple way to return a sequence of specified numbers.
- Its structure is as follows:

```
main.py  saved
1  for i in range(5):
2  |  print(i)
```

```
0
1
2
3
4
❖
```

# Random Number Generation (RNG)

- Random number generation is used in many projects, video games is one in particular.
  - Any time there is a chance for something to drop, random number generation is used to determine whether you get what you are looking for
- In Python, in order to use random number generation, you need to write a line of code to import the necessary library, import random

```
# Random  
import random  
rand = random.randint(1, 100)  
# This number is inclusive
```

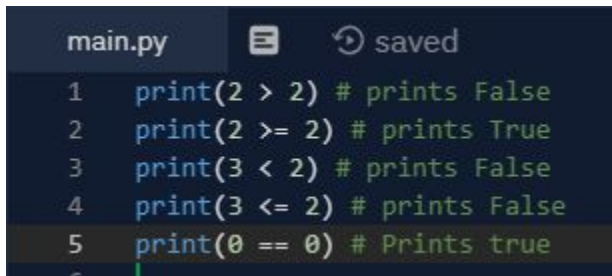
Day 3

# Agenda

- Logical Operators
- Boolean Expressions
- If Statements
- Updating Variables
- Introduction to Tic Tac Toe!

# Logical Operators

- Greater than, less than, and equal to are used in Python!
  - > is for greater than
  - >= is for greater than or equal to
  - < is for less than
  - <= is for less than or equal to
  - == is for equal to
- Expressions with these operators return a boolean value.



```
main.py  saved
1  print(2 > 2) # prints False
2  print(2 >= 2) # prints True
3  print(3 < 2) # prints False
4  print(3 <= 2) # prints False
5  print(0 == 0) # Prints true
```

# Logical Operators (cont.)

- You can combine various logical operators together by placing these between two expressions:
  - **and** - if both of the expressions are true, the combined expression is true. If not, it is false.
  - **or** - if **at least** one of the expressions is true, the combined expression is going to be true. If not, it is false.
  - **not** - turns the following expression into the opposite value of what it is
- Some examples:

```
main.py  saved
1  print(True and False) # prints out False
2  print(True and True) # prints out True
3  print(False and False) # prints out False
4
5  print(True or False) # prints out True
6  print(True or True) # prints out True
7  print(False or False) # prints out False
8
9  print(not True) # prints out False
10 print(not False) # prints out True
```

# Boolean Expressions

- Combining together a series of logical operators and variables gives you a boolean expression.
- You can use parentheses to dictate the order of the operations. Operations within parentheses will be performed first.
- Here are some examples. Try to predict what will be printed to the console (the answers are on the next slide):

```
main.py  saved
1  variable1 = True
2  variable2 = False
3  variable3 = True
4
5  print( (variable1 and variable2) or not variable3)
6  print( (variable1 and not variable2) or (variable2 and not variable3))
```

# Boolean Expressions (cont.)

```
False
```

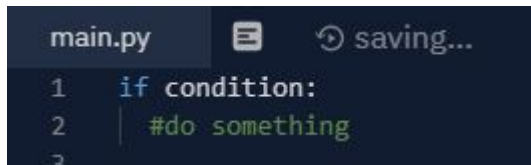
```
True
```





# If Statements

- If statements follow the simple logic of:
  - If **A** is true, then **B** will happen
- Since we are checking if something is true, it would only make sense that this is where **boolean values** are used
- The structure to an if statement is as follows:

A screenshot of a code editor window. The title bar shows 'main.py' and a 'saving...' status. The code is written in Python and shows an if statement. Line 1: 'if condition:', Line 2: '#do something'. Line 3 is partially visible and empty.

```
main.py  saving...
1  if condition:
2      #do something
3
```

# If Statements Examples

The following code prints:

```
main.py  saving...  
1  if True:  
2      print("line 1")  
3      print("line 2")
```

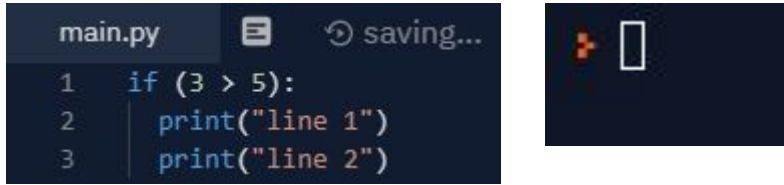
```
line 1  
line 2  
█
```

Notice how more than one line was printed from the if statement because they shared the same tabbing.

To have code run outside of the if statement, use the original tabbing.

# If Statements Examples (cont.)

Let's try another example:



The image shows a code editor window with a dark background. The file name 'main.py' is visible in the top left. To the right of the file name are icons for a menu and a 'saving...' status. The code in the editor is as follows:

```
1  if (3 > 5):  
2      print("line 1")  
3      print("line 2")
```

To the right of the code editor is a terminal window. It has a dark background and shows a single prompt character, a white vertical rectangle, indicating it is ready for input but has not yet executed any commands.

First, notice how nothing printed. When the condition is false, nothing runs.

Also notice how we can use logical operators from before to represent the boolean statement.

# Elif

- Elif means “if the previous condition was false, try this condition”.
- Here is how it works with if statements:

```
main.py  saving...
1  if condition:
2      #do something if the condition is true
3  elif otherCondition:
4      #do something if otherCondition is true, and condition was false
5
```

# Elif Examples

```
main.py  saving...
1  if True:
2      print("A")
3  elif True:
4      print("B")
5
```

```
A
[]
```

Notice how only the first print statement ran while the second was also true. This is because **elif only runs if the original if statement is false!**

The example below will run because the original if statement is false and the elif statement is true.

```
main.py  saving...
1  if False:
2      print("Will not print")
3  elif True:
4      print("Will print")
5
```

```
Will print
[]
```

# Else

- Else is used at the end of a series of conditions for when nothing else is true.
- Here is an example:

```
main.py  saving...
1  a = 10
2
3  if a < 3:
4      print("small")
5  elif a < 8:
6      print("medium")
7  else:
8      print("HUGE")
9
```

```
HUGE
[]
```

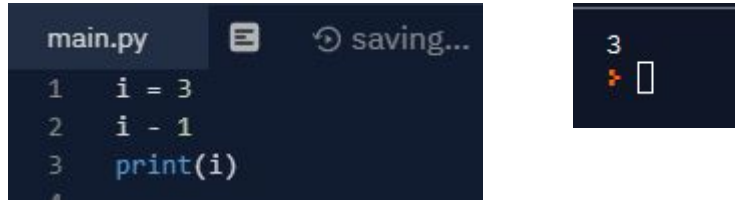
None of the previous conditions were true, so the else condition ran.

# Combined Example

```
main.py  [icon] saved
1  string_age = input("Enter your age: ")
2  number_age = int(string_age)
3
4  if number_age >= 18:
5      print("You can get your basic license.")
6  elif number_age < 18 and number_age >= 16:
7      print("You can get your probationary license.")
8  else:
9      print("You can't even drive!")
10
```

# Updating Variables -- Wrong Way

- Let's try changing the value of an already created variable
- A common thing new programmers try is shown below:



```
main.py  saving...  
1 i = 3  
2 i - 1  
3 print(i)  
4  
3
```

Notice how it stayed the same even though we subtracted by 1

This is because to update a variable, we must set the left side equal to the new variable. The code above simply calculates a value 2 that is not used anywhere and this ignored.



# Updating Variables -- Right Way

```
main.py  saving...  
1  i = 3  
2  i = i - 1  
3  print(i)
```

```
2  
□
```

By setting the variable `i` to this calculated value 2, we do update `i` this time

You can also do a shorthand version of this by typing `i -= 1`.

# Tic Tac Toe!

- <https://repl.it/@HealableHades1/TicTacToe-Goal>

EXTRA

# While Loops

- Essentially means “Do something until the condition is no longer true”
- The structure to a while loop is as follows:

```
main.py  [icon] saved
1  while condition:
2      #do something
```

# While Loop Example

```
main.py  [icon] saved
1  i = 3
2  while i > 0:
3      i = i - 1
4      print(i)
```

```
2
1
0
[icon]
```

Notice how, like if statements, loops are also based on indentation. Since two lines were indented below the while loop, they both were inside of the loop

Also notice how we correctly updated i

# Infinite While Loop

- Infinite while loops are used to run something forever, such as to run an application or game relying on input such as below:

```
main.py  saved
1  num1 = 1
2  num2 = 2
3
4  while True:
5      guess = input("What is " + str(num1) + " * " + str(num2))
6      if(int(guess) == num1 * num2):
7          print("Correct!")
8      else:
9          print("wrong :(")
10         num1 = num1 + 1
11         num2 = num2 + 1
12
```

```
What is 1 * 2 1
wrong :(
What is 2 * 3 6
Correct!
What is 3 * 4 12
Correct!
What is 4 * 5 20
Correct!
What is 5 * 6
```

Notice how the loop keeps going whether the answer was right or wrong

Also notice that it relies on input!

# Infinite Loop Warnings

- When using an infinite loop, you must have some sort of delay or else your program may crash
- Inputs are a good way to fix this because there is a delay for whenever a human enters something
- Too much memory is being used too fast when using an infinite loop without any delay such as below:

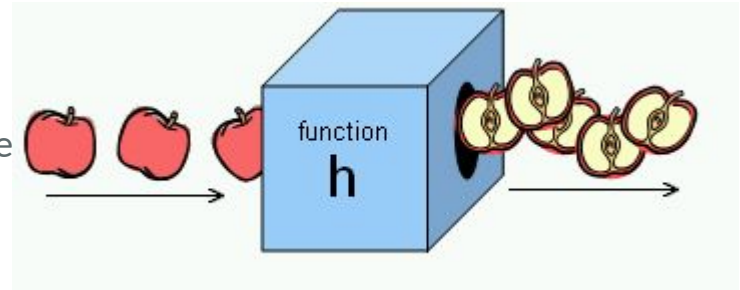
```
main.py  [icon] saved
1  i = 0
2
3  while True:
4      i = i + 1
5      print(i)
```

```
36727
36728
36729
36730
36731
36732
36733
36734
36735
```

```
36736
36737
36738
36739
36740
36741
36742
36743
36744
36745
36746
```

# Functions

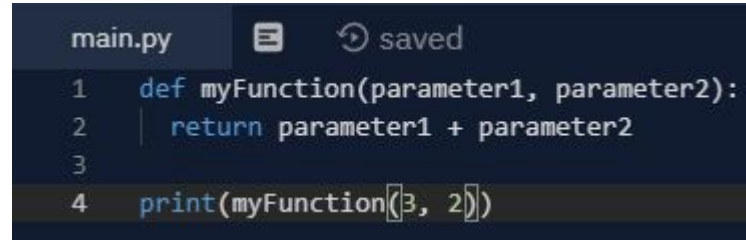
- A function is a section of reusable code that performs a single action.
- Functions take in values as input and output other values as output.
- We can make functions in our program in order to reduce the amount of code that we have to write.
- The function transforms the output into an input.
- We have already seen some functions.
  - The `print()` function takes in a string as input and outputs it to the console.
  - The `input()` function waits until the user inputs some text and gives it to the program as a string
- In addition to the default functions python gives us, we can also create our own functions





# Functions (cont.)

- We can make our own functions using the def keyword.



```
main.py  saved
1  def myFunction(parameter1, parameter2):
2      return parameter1 + parameter2
3
4  print(myFunction(3, 2))
```

- Here, we define a function myFunction(). It takes in two values and outputs the sum of these two values.
- We can have as many parameters as we want in python.
- The return statement determines what the output of the function will be.

# Functions (cont.)

```
main.py  [icon]  [icon] saved
1  def getAgeStatement(age):
2      number_age = int(age)
3      if number_age >= 18:
4          return "You can get your basic license."
5      elif number_age < 18 and number_age >= 16:
6          return "You can get your probationary license."
7      else:
8          return "You can't even drive!"
9
10
11 print(getAgeStatement("1231"))
12 print(getAgeStatement("32"))
13 print(getAgeStatement("3"))
14
```

```
You can get your basic license.
You can get your basic license.
You can't even drive!
```

# Escape Sequences

- Sometimes in a string, you need to type something out that requires you to use a character that the programming language has given another purpose
- In order to use these characters, programming languages have escape sequences.
- The most important escape sequences in Python are the following:
  - `\n` - This creates a new line
  - `\\` - This allows you to type a backslash without it being seen as an escape sequence
  - `\'` - This allows you to type a single quote without prematurely ending the string
  - `\"` - This allows you to type a double quote without prematurely ending the string

# Escape Sequences Example

```
1 print('He walked up to \'it\' to talk.\n\"Why are you doing this?\"')
2
3 print('\n~\\_(ツ)_/~-')
4 # Notice how we have to use two backslashes for the left arm to print
```

<https://LiquidSimultaneousLamp.suyog1.repl.run>

```
He walked up to 'it' to talk.
"Why are you doing this?"
```

```
~\\_(ツ)_/~-
```

