COL1000
Introduction to Programming

Priyanka Golia

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

Quick Quiz?

	Variable	Object	Statement
a= 3	✅	✅	✅
“Hello”	❌	✅	❌
print(“hello”)	❌	✅	✅

```
>>> number_1 = input("enter your number")
enter your number10
>>> number_2 = input("enter your number")
enter your number30
>>> print(number_1 + number_2)
1030
>>>
```

“Input” consider object are of type Str

```
>>> number_1 = input("enter your number")
enter your number10
>>> number_2 = input("enter your number")
enter your number30
>>> number_1 = int(number_1)
>>> number_2 = int(number_2)
>>> print(number_1 + number_2)
40
```

Str to Int conversion

```
>>> print(input("your name: "))
your name: priyanka
priyanka
>>> print(name = input("your name: "))
your name: priyanka
Traceback (most recent call last):
  File "<python-input-21>", line 1, in <module>
    print(name = input("your name: "))
    ~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: print() got an unexpected keyword argument 'name'
>>> print(x = 10)
Traceback (most recent call last):
  File "<python-input-22>", line 1, in <module>
    print(x = 10)
    ~~~~~^^^^^^^^
TypeError: print() got an unexpected keyword argument 'x'
>>> x = 10
>>> y = 20
>>> print( x + y)
30
>>>
```

Input inside Print is ok

Assignment op. inside print() is NOT ok

Key Concepts

Variables and Objects

Python builds objects using:

- Literals — directly written values:

  10, “hello”, [1,2,3]

- Operators — combine or manipulate other objects:

  3 + 4, “hi" + “there", [1] + [2]

- Functions — create or convert objects:

  int(“5”), list(range(3))

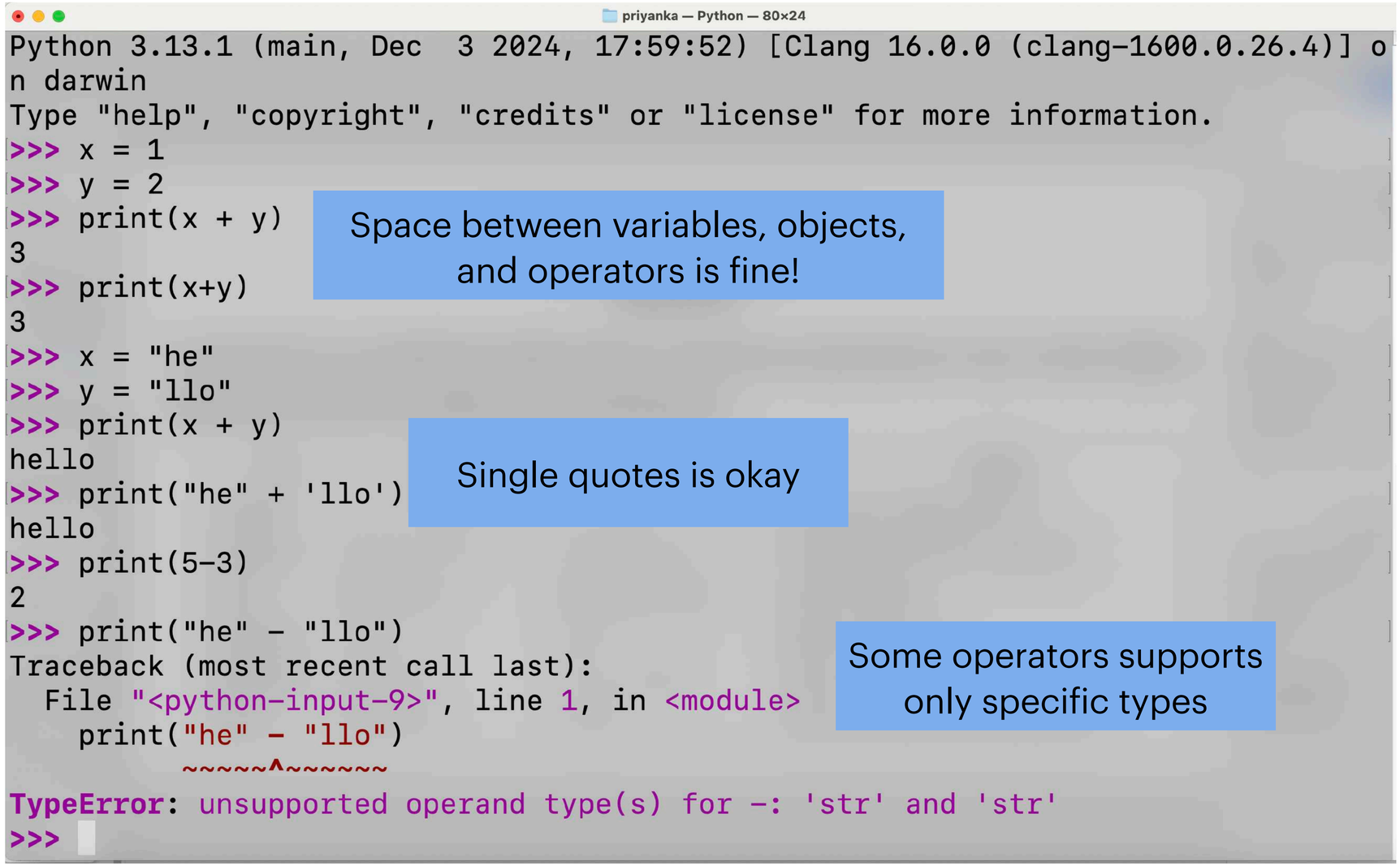
  int(“5”) — Object of type string is changed to object of type integer.

Key Concepts

Operators

- An operator is a symbol that tells Python to perform an operation on one or more objects.
- Operators create new objects from existing ones.

Type	Operator	Example	Meaning
Arithmetic	`+, -, *, /, //, %, **`	`3 + 4, 5 * 2`	Add, subtract, multiply, divide, etc.
Comparison	`==, !=, <, >, <=, >=`	`x == 10, x < y`	Check conditions (returns `True`/`False`)
Logical	`and, or, not`	`True and False`	Combine conditions
Assignment	`=, +=, -=`	`x += 1`	Update variable values
Membership	`in, not in`	`"a" in "cat"`	Check if value exists
Identity	`is, is not`	`x is y`	Check if two names refer to the same object


priyanka — Python — 80×24
Python 3.13.1 (main, Dec 3 2024, 17:59:52) [Clang 16.0.0 (clang-1600.0.26.4)] o
n darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1
>>> y = 2
>>> print(x + y)
3
>>> print(x+y)
3
>>> x = "he"
>>> y = "llo"
>>> print(x + y)
hello
>>> print("he" + 'llo')
hello
>>> print(5-3)
2
>>> print("he" - "llo")
Traceback (most recent call last):
File "<python-input-9>", line 1, in <module>
print("he" - "llo")
~~~~~^~~~~~~
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>>
Space between variables, objects, and operators is fine!
Single quotes is okay
Some operators supports only specific types
~~~~~

```
>>> print (7/3)
2.3333333333333335
>>> print(7//3)
2
>>> print(-7//3)
-3
>>> print(-7.0//3)
-3.0
>>> print(7.0//3)
2.0
>>> print(7//-3)
-3
>>>
```

// — integer division. Lower down to the closest integer. can handle negatives and “floats”!

```
>>> print(2 +"hello")
Traceback (most recent call last):
  File "<python-input-0>", line 1, in <module>
    print(2 +"hello")
          ~~^~~~~~~~
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> print(2 * "hello")
hellohello
>>>
```

```
>>> print(4 ** 2)
16
>>> print(4 * 2)
8
>>> print("hello" * 2)
hellohello
>>> print("hello" ** 2)
Traceback (most recent call last):
  File "<python-input-3>", line 1, in <module>
    print("hello" ** 2)
          ~~~~~~~~^^~~
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
>>>
```

** — power operator, $a**b$ is a^b

```
>>> x = 5
>>> x += 1   # x = x + 1
>>> print(x)
6
>>> y = 10
>>> y -= 1    # y = y - 1
>>> print(y)
9
>>> print("hello" == "hello")
True
>>> print("hello" == "heo")
False
>>> print("hello" == "HELLO")
False
>>> print(5  == 5)
True
>>> print(5  == 5.0)  == compares values, not types
True
```

```
>>> print ( 4 ** 2 *2)
32
>>> print (4 ** ( 2 * 2))
256
>>> print ( 4 + 2 * 4)
12
>>> print ( (4 + 2) * 4)
24
```

```
>>> print(-2 ** 2)
-4
>>> print((-2) ** 2)
4
```

** has higher precedence than Unary ops (-)

```
>>> print(10 // 3 /2)
1.5
>>> print(10 / 3 //2)
1.0
```

Check for precedence then left to right associativity

Operator Precedence.

Level	Examples
1. **Parentheses**	`(a + b) * c`
2. **Exponentiation**	`2 ** 3`
3. **Unary ops**	`-x, +x, not x`
4. Multiplication / Division	`*, /, //, %`
5. Addition / Subtraction	`+, -`
6. Bitwise shift operators	`<<, >>`
7. Bitwise operation	`&, ->, ^, ->, |`
8. Comparisons	`==, <, >`
9. Logical Op	`and, or`
10. Assignment	`=, +=, -=`

```
>>> print( 2 ** 3 ** 2)
512
>>> print( (2 ** 3) ** 2)
64
>>>
```

**, =, +=, -= — right to left associativity

Key Concepts

Variables and Objects

Python builds objects using:

- Literals — directly written values:

  10, “hello”, [1,2,3]

- Operators — combine or manipulate other objects:

  3 + 4, “hi" + “there", [1] + [2]

- Functions — create or convert objects:

  int(“5”), list(range(3))

  int(“5”) — Object of type string is changed to object of type integer.

Key Concepts

Functions

Already seen: print, input

A function is a named block of code that you can reuse.

Some functions are built in— print, input, min, max..

```
>>> print("hello", " there!")
hello  there!
>>> min(4,3,1,0,-1)
-1
>>> max(4,3,1,0,-1)
4
>>>
```

function_name(parameters)
Parameters — enclosed by parentheses, separated by comma

```
>>> minimum = min(4,3,1,0,-1)
>>> print(minimum)
-1
>>>
```

Could assign “result” of the function to a variable!

Key Concepts

Functions — Importing Modules

- In Python, import lets you bring in ready-made code written by others — so you don't have to write everything from scratch!
- For example — built-in module called math that gives you access to many useful math-related functions and constants, like **sqrt, floor, ceil, pi, abs** etc.

```
Import Math
```

This tells, " I want to use the **Math** module"

```
import math
# math.sqrt is a function that computes square root
print(math.sqrt(16))     # 4.0
```

```
module_name.function_name(arguments)
```

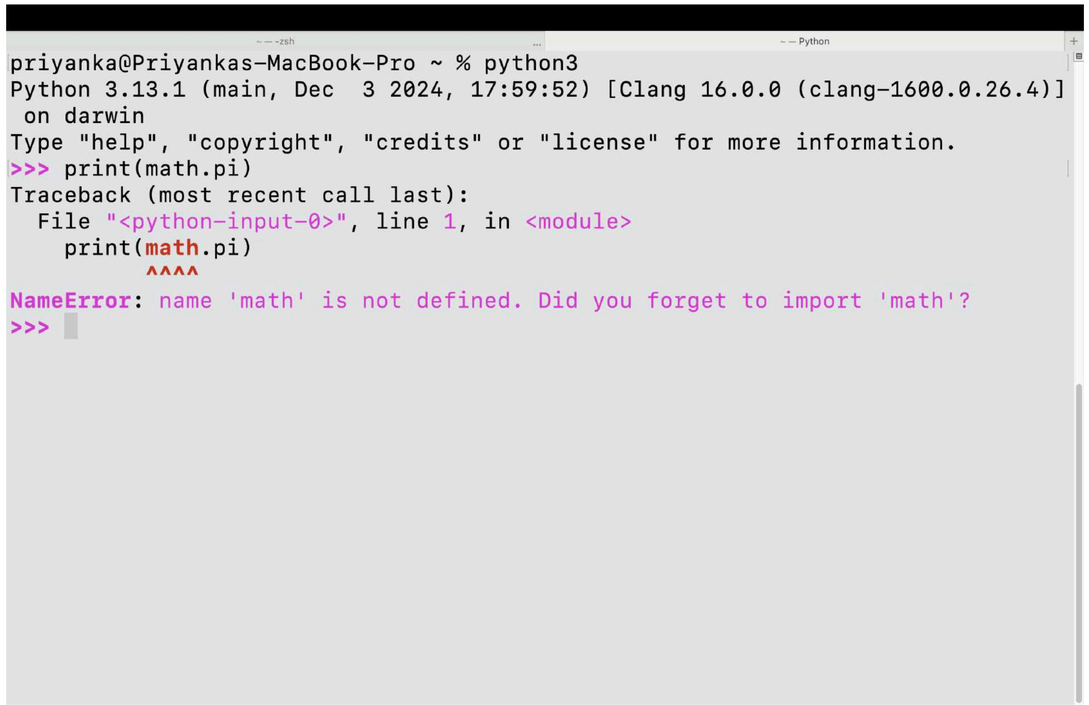
Key Concepts

Functions — Importing Modules

Python has many built-in modules with useful functions, but you need to import them first:

```
>>> print(math.pi)
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
```



The correct way to use modules:

```
>>> import math
>>> print(math.pi)
```

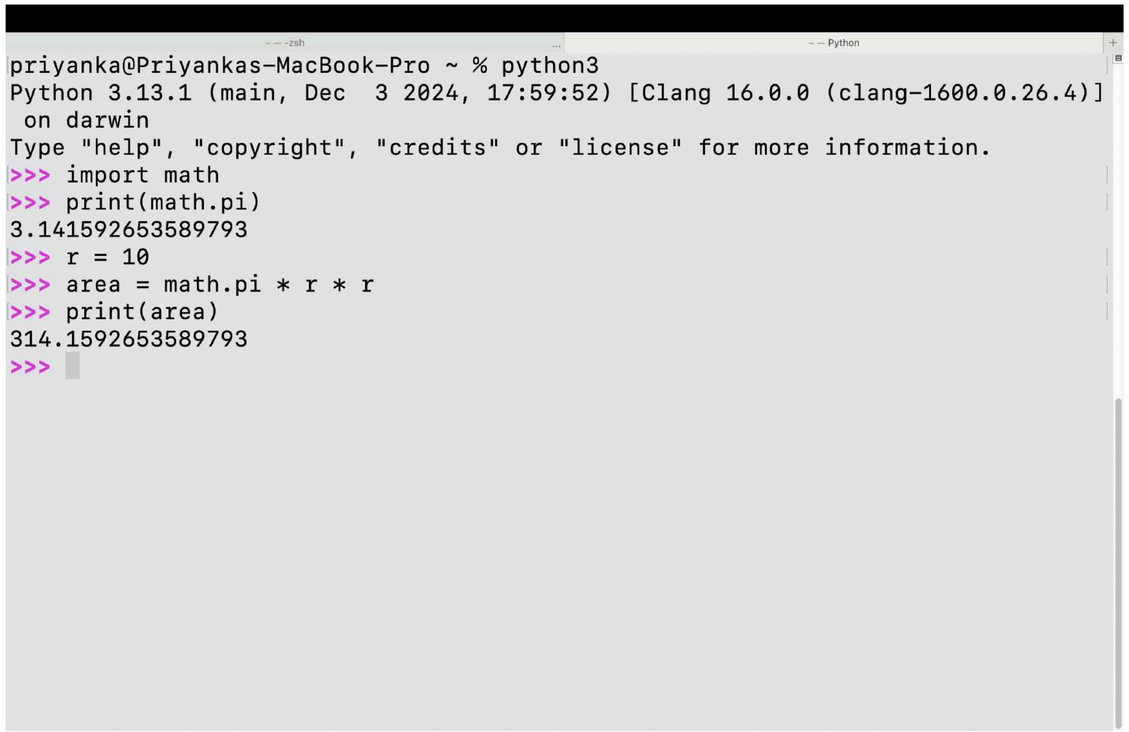
```
3.141592653589793
```

Key Concepts

Functions — Importing Modules

Python's `math` module gives you access to mathematical constants and functions:

```
>>> import math
>>> print(math.pi)
3.141592653589793
>>> r = 10
>>> area = math.pi * r * r
```



Is π exactly equal to 22/7?

```
>>> math.pi == 22/7
False
```

The math module provides more precise values than common approximations!

Key Concepts

Functions

Indentation (tabs or spaces) is used to define blocks of code

```
# Without defining a new function
print("Hello Priyanka!")
print("Hello Rahul!")
print("Hello Meena!”)
```

“:” is important

Function definition!

Colon (:) tells that a block of code is coming next, and it belongs to this statement.

```
def say_hello(name):
    print(“Hello “, name)

say_hello(“Priya")
say_hello("Rahul")
say_hello("Meena")
```

Function call!

Same output!

Common Python Errors

Python will tell you when something's wrong:

```
>>> Please print "IITD students are awesome!".
```

```
File "<stdin>", line 1
Please print "IITD students are awesome!".
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

Pro tip: Python's error messages are helpful! They often suggest how to fix the problem.

Common beginner errors:

- Missing parentheses in function calls
- Incorrect indentation
- Using undefined variables