



# Introduction to Python

## Day 2

RIEL SOVANNDOEUR




# VARIABLES AND DATA TYPES



(mostly strings for now)



# OBJECTIVES

- Understand how to assign and use variables
  - Learn Python variable naming restrictions and conventions
  - Learn and use some of the different data types available in Python
  - Learn why Python is called a dynamically-typed language
  - Understand how to convert data types
- 



# VARIABLE ASSIGNMENT

A variable in Python is like a variable in mathematics: it is a named symbol that holds a value.

```
x = 100
khaleesi_mother_of_dragons = 1
print(khaleesi_mother_of_dragons + x)
101
```

Variables are always **assigned** with the variable name on the left and the value on the right of the equals sign.

Variables must be assigned before they can be used.



# VARIABLE ASSIGNMENT

(continued)

Variables can be:

- assigned to other variables
- reassigned at any time
- assigned at the same time as other variables

```
python_is_awesome = 100  
just_another_var = python_is_awesome  
python_is_awesome = 1337  
all, at, once = 5, 10, 15
```

# Naming Restrictions

In Python, you can name your variables whatever you want, with some restrictions:

1. Variables must start with a letter or underscore

`_cats` ✓ `2cats` ✗

2. The rest of the name must consist of letters, numbers, or underscores

`cats2` ✓ `hey@you` ✗

3. Names are case-sensitive

`CATS != Cats`     `Cats != cats`

# DATA TYPES

In any assignment, the assigned value must always be a valid data type.

Python data types include:

data type	description
bool	True or False values
int	an integer (1, 2, 3)
str	(string) a sequence of Unicode characters, e.g. "Colt" or "程序设计"
list	an ordered sequence of values of other data types, e.g. [1, 2, 3] or ["a", "b", "c"]
dict	a collection of key: values, e.g. { "first_name": "Colt", "last_name": "Steele" }

Among others!

# Dynamic Typing

Python is highly flexible about reassigning variables to different types:

```
awesomeness = True
print(awesomeness) # True

awesomeness = "a dog"
print(awesomeness) # a dog

awesomeness = None
print(awesomeness) # None

awesomeness = 22 / 7
print(awesomeness) # 3.142857142857143
```

We call this **dynamic typing**, since variables can change types readily.

Other languages, such as C++, are **statically-typed**, and variables are stuck with their originally-assigned type:

```
int not_awesomeness = 5;
not_awesomeness = "cool"; // !Error
```



# Declaring Strings

String literals in Python can be declared with either single or double quotes.

```
my_other_str = 'a hat'  
my_str = "a cat"
```

Either one is perfectly fine; but make sure you stick to the same convention throughout the same file.

# String Escape Characters

In Python there are also "escape characters", which are "metacharacters" - they get interpreted by Python to do something special:

```
new_line = "hello \n world"

print(new_line)
# hello
# world
```

All escape characters start with a backslash "\".

You can do advanced things like include hexadecimal characters with "\x"

```
hexadecimal = "\x41\x42\x43" # "abc"
```

# String Concatenation

Concatenation is combining multiple strings together. In Python you can do this simply with the "+" operator.

```
str_one = "your"  
str_two = "face"  
str_three = str_one + " " + str_two # your face
```

You can also use the "+=" operator!

```
str_one = "ice"  
str_one += " cream"  
str_one # ice cream
```

# Formatting Strings

There are also several ways to format strings in Python to **interpolate** variables.

The new way (new in Python 3.6+) => **F-Strings**

```
x = 10
formatted = f"I've told you {x} times already!"
```

The tried-and-true way (Python 2 -> 3.5) => **.format method**

```
x = 10
formatted = "I've told you {} times already!".format(x)
```

The old way => **% operator** (deprecated)

```
x = 10
formatted = "I've told you %d times already!" % (x)
```

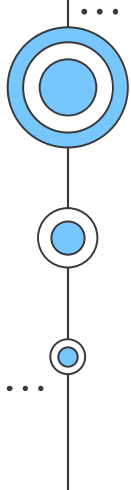
# Converting Data Types

In string interpolation, data types are implicitly converted into string form (more on this later in OOP).

You can also explicitly convert variables by using the name of the builtin type as a function (more on functions later):

```
decimal = 12.56345634534  
integer = int(decimal) # 12
```

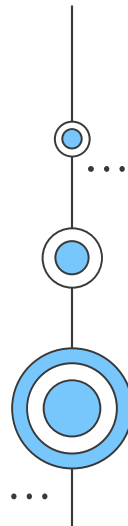
```
my_list = [1, 2, 3]  
my_list_as_a_string = str(my_list) # "[1, 2, 3]"
```

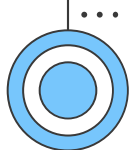


# QUIZ

# TIME

It's super quick, I promise





## Exercise 2: Hello

*(9 Lines)*

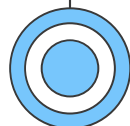
Write a program that asks the user to enter his or her name. The program should respond with a message that says hello to the user, using his or her name.

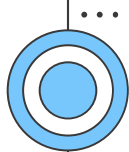
## Exercise 3: Area of a Room

*(Solved—13 Lines)*

Write a program that asks the user to enter the width and length of a room. Once the values have been read, your program should compute and display the area of the room. The length and the width will be entered as floating point numbers. Include units in your prompt and output message; either feet or meters, depending on which unit you are more comfortable working with.

...





## Exercise 4: Area of a Field

*(Solved—15 Lines)*

Create a program that reads the length and width of a farmer's field from the user in feet. Display the area of the field in acres.

Hint: There are 43,560 square feet in an acre.

## Exercise 5: Sum of the First $n$ positive Integers

*(Solved—12 Lines)*

Write a program that reads a positive integer,  $n$ , from the user and then displays the sum of all of the integers from 1 to  $n$ . The sum of the first  $n$  positive integers can be computed using the formula:

$$\text{sum} = \frac{(n)(n + 1)}{2}$$

