# Python

A quickstart into the key concepts of programming Variables & operators

# Key concepts in programming

- Variables (integers, strings, dates, etc.)
- Flow control (if then, loop, etc.)
- Functions (list of steps the code will follow)

# **Variables**

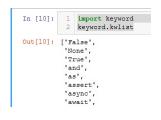
variables.ipynb

#### **Variables**

- Variables are placeholders for *locations in memory*.
  - Variables are names for values
  - Created by use no declaration necessary
- Variables always have a type
  - Variables only have data types after you use them
  - Use the type function to determine variable type
- Variables have a name
  - Python is case sensitive.
    - myVar is different from Myvar
    - Tip: avoid using names that differ only by case.

### What variable names are legal?

- Choose meaningful names
- No leading numbers, no spaces
- lowercase, with words separated by underscores as necessary to improve readability (same for function naming)
- Don't use Python keywords



#### x = x + 1

- $\bullet x = x + 1$
- Evaluate the value on the right hand side of the equal sign
  - need to know what the current value of x
  - Ex. x = 7, then x + 1 evaluates to 8
- Assign this value (i.e. 8) to the variable name shown on the left hand side x.
- it is a quite a common operation to increase a variable x by some fixed amount c, we can write
  - $\bullet$  x = x + c
  - x += c
  - Note that the order of + and = matters

### Python Variables Are References

- Variables must be created (assigned a value) before they can be used
- A variable is created through assignment:

```
x = 4
```

- What happens?
  - · Python creates the object 4
    - Everything in Python is an object, this object is stored somewhere in memory.
  - Python binds a name to the object. x is a reference to the object.
- · Consequences:
  - No need to "declare" the variable
  - No need to require the variable to always point to information of the same type.
  - dynamically typed: variable names can point to objects of any type.

```
In [1]: x = 1 \# x \text{ is an integer}

x = \text{'hello'} \# now x \text{ is a string}

x = [1, 2, 3] \# now x \text{ is a list}
```

### Everything is an object

- In Python, everything is an object:
  - Some associated functionality (methods) and metadata (attributes).
  - These methods and attributes are accessed via the dot(.) syntax.
  - Use type to get information on the class
  - Use dir to get an overview on the methods
- File: check\_variable\_object.py

#### dir

• See all the methods that are bound to an object

```
dir('')
```

- Many of the names in the list start and end with two underscores (dunder), like \_\_add\_\_. These are all associated with methods and pieces of data used internally by the Python interpreter.
- The remaining entries in the list are all user-level methods.
- Object notation

object.method(parameters)

## Immutable vs Mutable Objects

- In Python, there are two types of objects:
  - Immutable objects can't be changed.
  - Mutable objects can be changed.

Туре	Immutable?
Int	Yes
Float	Yes
Bool	Yes
Complex	Yes
Tuple	Yes
Str	Yes
List	No
Set	No
Dict	No

# Immutable vs Mutable Objects

```
In [1]:
               x = 1 \# x \text{ is an integer}
               x = 'hello' # now x is a string
               x = [1, 2, 3] # now x is a list
In [2]:
              x = [1, 2, 3]
In [3]:
              print(y)
               [1, 2, 3]
In [4]:
              x.append(4) # append 4 to the list pointed to by x
              print(y) # y's list is modified as well!
[1, 2, 3, 4]
In [5]:
             x = 'something else'
               print(y) # y is unchanged
[1, 2, 3, 4]
```

File: variables\_are\_pointers.py

Check: https://realpython.com/pointers-in-python/

# Operators

operators.ipynb

## Arithmetic Operations

Operator	Name	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference of a and b
a * b	Multiplication	Product of a and b
a / b	True division	Quotient of a and b
a // b	Floor division	Quotient of a and b, removing fractional parts
a % b	Modulus	Remainder after division of a by b
a ** b	Exponentiation	a raised to the power of b
-a	Negation	The negative of a
+a	Unary plus	a unchanged (rarely used)

# **Assignment Operations**

- A = value (regular assignment)
- a #= b is equivalent to a = a # b

## **Comparison Operations**

a == b	a equal to b
a != b	a not equal to b
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

## Boolean operator

- and, or, not
- A good general rule is to always use parentheses when mixing and and or in the same condition.
- Different from the bitwise operator! (&,  $\mid$ ,  $\sim$ )

$$x = 4$$
 $(x < 6)$  and  $(x > 2)$ 
 $2 < x < 6$ 

# Bitwise operator

- bitwise operators only make sense in terms of the binary representation
- Use built-in bin function

Operator	Name	Description
a & b	Bitwise AND	Bits defined in both a and b
a   b	Bitwise OR	Bits defined in a or b or both
a ^ b	Bitwise XOR	Bits defined in a or b but not both
a << b	Bit shift left	Shift bits of a left by b units
a >> b	Bit shift right	Shift bits of a right by b units
~a	Bitwise NOT	Bitwise negation of a