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



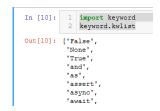
Variables

- 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





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
 - dynamically typed: variable names can point to objects of any type.
 - No need to require the variable to always point to information of the same type.

```
x = 1 # x is an integer
x = 'hello' # now x is a string
x = [1, 2, 3] # now x 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



Python Variables Are Pointers

```
x = 1 # x is an integer
x = 'hello' # now x is a string
x = [1, 2, 3] # now x is a list
x = [1, 2, 3]
y = x
print(y)
[1, 2, 3]
File: variables are pointers.py
```

Check: https://jakevdp.github.io/WhirlwindTourOfPython/03-semantics-variables.html



dir

- The dir() function returns all properties and methods of the specified object, without the values.
- 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.
 - dir() is supplied primarily as a convenience for use at an interactive prompt, it tries to supply an interesting set of names more than it tries to supply a rigorously or consistently defined set of names, and its detailed behavior may change across releases.
- Object notation

object.method(parameters)



id() function

- What is the number returned from the function?
 - It is "an integer (or long integer) which is guaranteed to be unique and constant for this object during its lifetime." (Python Standard Library Built-in Functions)
- Is it similar to memory addresses in C?
 - In CPython, this will be the address of the object.
 - This identity is unique to the Python interpreter, and should not be considered an actual physical address in memory. (Justin Bois http://justinbois.github.io/bootcamp/2022/lessons/l05_lists_and_tuples.html)

$$x = x + 1$$

- $\bullet x = x + 1$
- 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



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! (&, |, ~)

$$x = 4$$

(x < 6) **and** (x > 2)
2 < x < 6



The is operator

- Compares two objects and determines whether they are exactly the same. The is operator evaluates to true if the variables on either side of the operator point to the same object and false otherwise.
- The == operator is used when the *values* of two operands are equal, then the condition becomes true.

```
a = 8.8
b = 8.8
print('a == b', a==b)
print('a is b', a is b)
print('id(a): ', id(a))
print('id(b): ', id(b))
a == b True
a is b False
id(a): 1772400854512
id(b): 1772400854256
```



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

