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

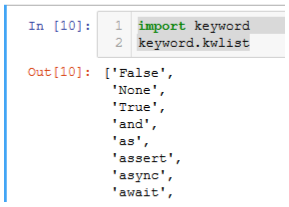
variables.ipynb

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



```
In [10]:  import keyword
          keyword.kwlist

Out[10]: ['False',
          'None',
          'True',
          'and',
          'as',
          'assert',
          'async',
          'await',
```

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

Python Variables Are Pointers

```
In [1]:         x = 1 # x 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]
                y = x
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

x = x + 1

- `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
  - `x = x + c`
  - `x += c`
  - Note that the order of + and = matters

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)
```

Objects and values

- `a = 'banana'`
- `b = 'banana'`
- a and b both refer to a string, but we don't know whether they refer to the same string.
- To check whether two variables refer to the same object, you can use the `is` operator.
  - `>>> a is b`
  - `True`
  - In this example, Python only created one string object, and both a and b refer to it.
- Create two lists, you get two objects:
- `a = [1, 2, 3]`
- `b = [1, 2, 3]`
- `>>> a is b`
- `False`
- In this case we would say that the two lists are equivalent, because they have the same elements, but not identical, because they are not the same object. If two objects are identical, they are also equivalent, but if they are equivalent, they are not necessarily identical.

https://eng.libretexts.org/Bookshelves/Computer_Science/Book:_Python_for_Everybody_(Severance)

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

```
a+= b     a -= b   a *= b    a /= b
a //= b   a %= b   a **= b   a &= b
a |= b    a ^= b   a <<= b   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
```

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