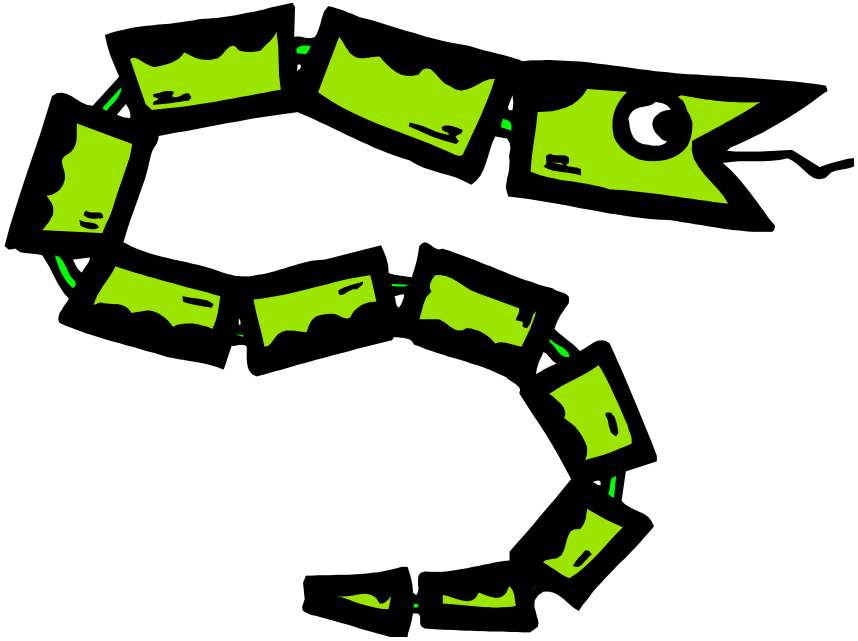
The Basics



A Code Sample (in IDLE)

```
x = 34 - 23                 # A comment.
y = “Hello”                 # Another one.
z = 3.45
if z == 3.45 or y == “Hello”:
    x = x + 1
    y = y + “ World”        # String concat.
print x
print y
```

Enough to Understand the Code

- **Indentation matters to code meaning**
  - Block structure indicated by indentation
- **First assignment to a variable creates it**
  - Variable types don't need to be declared.
  - Python figures out the variable types on its own.
- **Assignment is `=` and comparison is `==`**
- **For numbers `+ - * / %` are as expected**
  - Special use of `+` for string concatenation and `%` for string formatting (as in C's printf)
- **Logical operators are words (`and, or, not`) *not* symbols**
- **The basic printing command is `print`**

Basic Datatypes

- **Integers (default for numbers)**

```
z = 5 / 2  # Answer 2, integer division
```

- **Floats**

```
x = 3.456
```

- **Strings**
  - Can use “” or ‘ to specify with “abc” == ‘abc’
  - Unmatched can occur within the string: “matt’s”
  - Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them: “““a‘b“c”””

Whitespace

Whitespace is meaningful in Python: especially indentation and placement of newlines

- Use a newline to end a line of code
  - Use `\` when must go to next line prematurely
- No braces `{}` to mark blocks of code, use *consistent* indentation instead
  - First line with *less* indentation is outside of the block
  - First line with *more* indentation starts a nested block
- Colons start of a new block in many constructs, e.g. function definitions, then clauses

Comments

- Start comments with #, rest of line is ignored
- Can include a “documentation string” as the first line of a new function or class you define
- Development environments, debugger, and other tools use it: it’s good style to include one

```
def fact(n):
    """fact(n) assumes n is a positive
    integer and returns facorial of n."""
    assert(n>0)
    return 1 if n==1 else n*fact(n-1)
```

Assignment

- *Binding a variable* in Python means setting a *name* to hold a *reference* to some *object*
  - *Assignment creates references, not copies*
- Names in Python do not have an intrinsic type, objects have types
  - Python determines the type of the reference automatically based on what data is assigned to it
- You create a name the first time it appears on the left side of an assignment expression:

```
x = 3
```

- A reference is deleted via garbage collection after any names bound to it have passed out of scope
- Python uses *reference semantics* (more later)

Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

```
bob   Bob   _bob   _2_bob_   bob_2   BoB
```

- There are some reserved words:

```
and, assert, break, class, continue,
def, del, elif, else, except, exec,
finally, for, from, global, if,
import, in, is, lambda, not, or,
pass, print, raise, return, try,
while
```

Naming conventions

The Python community has these recommend-ed naming conventions

- **joined_lower** for functions, methods and, attributes
- **joined_lower** or **ALL_CAPS** for constants
- **StudlyCaps** for classes
- **camelCase** only to conform to pre-existing conventions
- Attributes: interface, _internal, __private

Assignment

- You can assign to multiple names at the same time

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

This makes it easy to swap values

```
>>> x, y = y, x
```

- Assignments can be chained

```
>>> a = b = x = 2
```

Accessing Non-Existent Name

Accessing a name before it's been properly created (by placing it on the left side of an assignment), raises an error

```
>>> y

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```