PIC 16, Winter 2018

Lecture 4M: IO

Monday, January 29, 2018

Matt Haberland

Announcements

- Assignment 3W due
- As we get further into the quarter, I will be less likely to give credit for questions that are easily researched’
- Erratum in 3F: if you define `__getitem__`, your class gets a default iterator and works with `for` loops automatically.

Intended Learning Outcomes

By the end of lecture, students are intended to be able to:

- work with the binary and hexadecimal number systems,
- describe how the `with`…`as` construct works and employ it when appropriate, and
- use the string methods `split`, `join`, and `format`.

Activities

- Finish assignment 3W
- Work on assignment 3F
- Start assignment 4M (optional)

A Bit About Binary

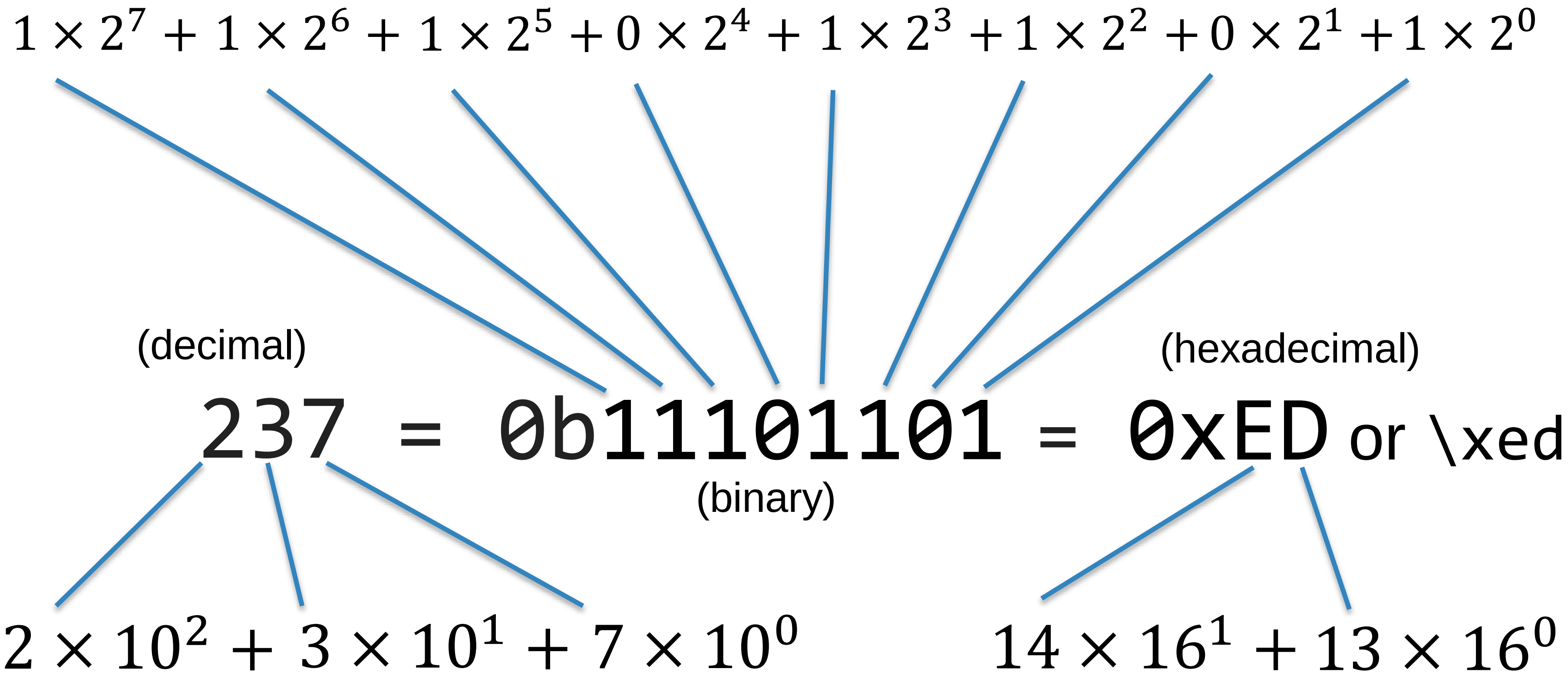
- 1 bit = (a one or a zero)
- 1 byte = 8 bits
- 1 kilobyte = 10^3 or 2^{10} bytes
- 1 megabyte = 10^6 or 2^{20} bytes
- 1 gigabyte = 10^9 or 2^{30} bytes
- etc…

Data Transfer / Disk Storage

Random Access Memory

```
E8\_E,trArdWX9f^zo)x0Q&B34<A=FE$jERlqqh+*rVqlIC%J#JQe$*_@80
(V[fg8uZfGD.s?PT.*\8;8T\S%!rD<YwZA%LvV'8"rz).eLcJ&g%SHJ]PA0
rk`q?.G4:BQ'[2}|b;TmITKAN\'Y8{_inJSekcfC3o@t"qkB\D*?K?~-
C.fzXbY"5)j_Ocy*j;!{36v|w5Y3Zhv"&:%u![lH}=sY^O_\&K3,VlIb`44
Np?w@UjtEl4)<vtyet0mv=WG:=VB,I'Yn~{n&%nSqpCkWJR`edfqQjQS%S5
3LUVfj^2JsaHC&`RVxx&Jk1]PMlP(r:c)_,J3{#%^vyQwffU.M[`mO,}nN;
:<:_j#?{`M^`]g>TL'w8|Ql+R@2]DWtVA~'Y}g$t&E0xEekw<)`3JG%)74+
izwVA4/Tdf;[4;_S[K7_[QNMp`+r&ZkVg=w12Fi4&LITg88T40>3+FqMn0R
h')=_`'@x:'O$2@C,@bFkIsVSF[[Idy!>tURQDmaHo\!qedocBoCe:8)@8E
FZ)}VC5R~C<OX%LqPjPZ5CD(_0G9}3$M4vY(b:DIZ'spTq*aKh/yg2gPvy0
r%BZhDju)=z1w(/yXiw.7cpFq^Sr=>:u!h@9?{xxKM?/yHZ.Sd(ESDsm>VJ
`a|W&Nd3}e_aeEr6R`]8i^;O]nVNmajo`EF$fIQ>DF.%M5Bgql2+gw4IBOH
Rih)Yl@:?O9HeZT%3Oi3"5Jb.k&~[C\9{Q7P.X/plb}1??Ity*nyFD]&rdB
_v^/.D7LnR]~W73l#cb$MzeO:(x:k{uK3et,nRy?g"@bT6hKbqOOc?[:>3D
q`)bL*M,IUXwI.iigI{9eeQYK0toti<A,SOO'Pc:Ij$fyYuWoUmb$E;q]F^
=%}::Jf:&wk&`"/=UGF;b*fz;s~fyl)X=~tnR=&xn.pH4o:Co!>5K\)bQ^j
Z_R%Ht}E}.:o{Tnj;Kou\Bj"}<DQ&9JYc\LRB8Q]/[iEwk/Kc&fM0Ry8~_^
```

A Bit About Binary



$$1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

(decimal)

(hexadecimal)

`237 = 0b11101101 = 0xED` or `\xed`

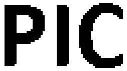
(binary)

$$2 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$$

$$14 \times 16^1 + 13 \times 16^0$$

In Hexadecimal we use numerals 0-9 and A = 10, B = 11, C = 12, D = 13, E = 14, F = 15

Binary Data in Python



Monochrome image in Windows bitmap file format.
69 px × 39 px, 530 bytes

Binary data is loaded as a string. Hexadecimal notation is used unless byte value is within the range of printable ASCII characters.

```
BM\x12\x02\x00\x00\x00\x00\x00\x00>\x00\x00\x00(\x00\x00\x00E\x00\x00\x00'\x00\x00\x00\x01\x00\x01\x00\
x00\x00\x00\x00\xd4\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\xff\xff\xff\x00\xff\xff\xff\xff\xff\xff\xff\xff\xf8\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xf
f\xf8\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xf8\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xf8\x
00\x00\x00\xff\xff\xff\xff\xff\xff\xc0\x0f\xf8\x00\x00\x00\xf8?\xff\xfe\x0f\xff\x80\x01\xf8\x00\x00\x00
\xf8\x1f\xff\xfe\x07\xfe\x00\x00\xf8\x00\x00\x00\xf8\x1f\xff\xfe\x07\xfc\x00\x00\xf8\x00\x00\x00\xf8\x1
f\xff\xfe\x07\xf8\x00\x00\xf8\x00\x00\x00\xf8\x1f\xff\xfe\x07\xf0\x07\xf0\xf8\x00\x00\x00\xf8\x1f\xff\x
fe\x07\xf0\x0f\xfc\xf8\x00\x00\x00\xf8\x1f\xff\xfe\x07\xe0\x1f\xff\xf8\x00\x00\x00\xf8\x1f\xff\xfe\x07\
xe0?\xff\xf8\x00\x00\x00\xf8\x1f\xff\xfe\x07\xe0\x7f\xff\xf8\x00\x00\x00\xf8\x1f\xff\xfe\x07\xc0\x7f\xf
f\xf8\x00\x00\x00\xf8\x1f\xff\xfe\x07\xc0\x7f\xff\xf8\x00\x00\x00\xf8\x00?\xfe\x07\xc0\x7f\xff\xf8\x00\
x00\x00\xf8\x00\x0f\xfe\x07\xc0\x7f\xff\xf8\x00\x00\x00\xf8\x00\x03\xfe\x07\xc0\xff\xff\xf8\x00\x00\x00
\xf8\x00\x01\xfe\x07\xc0\xff\xff\xf8\x00\x00\x00\xf8\x00\x00\xfe\x07\xc0\xff\xff\xf8\x00\x00\x00\xf8\x1
f\x00\xfe\x07\xc0\xff\xff\xf8\x00\x00\x00\xf8\x1f\xc0~\x07\xc0\x7f\xff\xf8\x00\x00\x00\xf8\x1f\xe0~\x07
\xc0\x7f\xff\xf8\x00\x00\x00\xf8\x1f\xe0~\x07\xc0\x7f\xff\xf8\x00\x00\x00\xf8\x1f\xe0~\x07\xe0\x7f\xff\
xf8\x00\x00\x00\xf8\x1f\xe0~\x07\xe0?\xff\xf8\x00\x00\x00\xf8\x1f\xe0~\x07\xe0?\xff\xf8\x00\x00\x00\xf8
\x1f\xc0~\x07\xf0\x1f\xfc\xf8\x00\x00\x00\xf8\x1f\x80~\x07\xf0\x0f\xf8\xf8\x00\x00\x00\xf8\x00\x00\xfe\
x07\xf8\x07\xe0\xf8\x00\x00\x00\xf8\x00\x00\xfe\x07\xfc\x00\x00\xf8\x00\x00\x00\xf8\x00\x01\xfe\x07\xfe
\x00\x00\xf8\x00\x00\x00\xf8\x00\x07\xfe\x07\xff\x00\x01\xf8\x00\x00\x00\xf8\x00?\xfe\x0f\xff\x80\x03\x
f8\x00\x00\x00\xff\xff\xff\xff\xff\xff\xf0\x1f\xf8\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xf8\x00\
x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xf8\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xf8\x00\x00\x00
```

with…as

```
with open(filename,'r') as f:
    t = f.read()
```

is sort of like:

```
f = open(fname,'r')
try:
    t = f.read()
finally:
    f.close()
```

The purpose of using this syntax is to ensure that the file gets closed *no matter what*.

with…as

Technically:

```
with expression as a:
    stuff to do
```

gets evaluated like:

```
a = expression
a.__enter__()
t, value, traceback = None, None, None
try:
    stuff to do
except Exception as e:
    t = type(e); value = str(e); traceback = traceback object
finally:
    a.__exit__(t, value, traceback)
```

Expression must return an object with two methods: `__enter__`, and `__exit__`. Custom classes can work with this construct by defining these methods.

with…as

```
class WithAs:
    def __init__(self):
        print "Initializing..."

    def __enter__(self):
        print "Entering...“
        return self

    def __exit__(self, typ, value, traceback):
        print typ
        print value
        print traceback
        print "Exiting..."

    def do_something(self):

        print "Doing something..."

def f():
    return WithAs()

with f() as w:
    w.do_something()
```

```
Initializing...
Entering...
Doing something...
None
None
None
Exiting...
```

with…as

```
class WithAs:
    def __init__(self):
        print "Initializing..."

    def __enter__(self):
        print "Entering...“
        return self

    def __exit__(self, typ, value, traceback):
        print typ
        print value
        print traceback
        print "Exiting..."

    def do_something(self):
        raise Exception("Message")
        print "Doing something..."

def f():
    return WithAs()

with f() as w:
    w.do_something()
```

```
Initializing...
Entering...
<type 'exceptions.Exception'>
Message
<traceback object at 0x0D7F73A0>
Exiting...
```

Also, traceback prints to console.

split

```
s = "These are separate words.\nHere is a new line."
print s.split()
```

```
['These', 'are', 'separate', 'words.', 'Here', 'is', 'a', 'new', 'line.']
```

```
s = "These are separate words.\nHere is a new line."
print s.split(" ")
```

```
['These', 'are', 'separate', 'words.\nHere', 'is', 'a', 'new', 'line.']
```

Default splits at any whitespace; " " doesn't.

```
s = "These are separate words.\nHere is a new line."
print s.split("ar")
```

```
['These ', 'e sep', 'ate words.\nHere is a new line.']
```

You can split with any substring; that substring is removed.

join

```
l = ['These', 'are', 'separate', 'words.']
print " ".join(l)
```

```
These are separate words.
```

```
l = ['These', 'are', 'separate', 'words.']
print ",\n".join(l)
```

```
These,
are,
separate,
words.
```

You can join with any string.

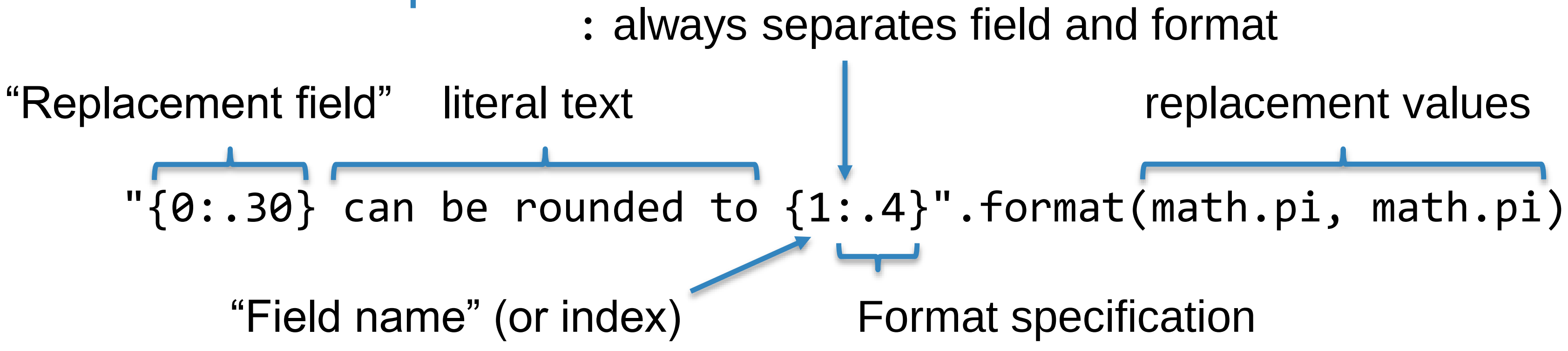
`split` and `join` are inverses

```
s == string.join(string.split(s, sep), sep)
```

They can also be invoked as functions (from the `string` module)

`format`

Usually I don’t like using `format`, but here are two very useful examples.



`:` always separates field and format

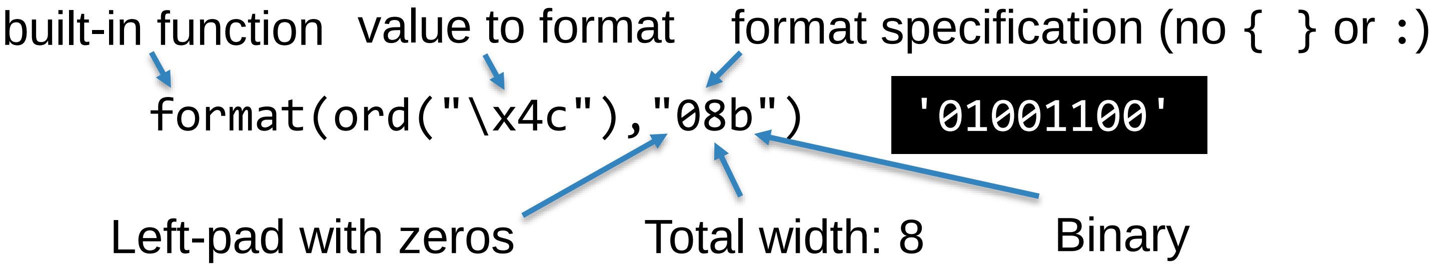
“Replacement field” literal text replacement values

```
"{0:.30} can be rounded to {1:.4}".format(math.pi, math.pi)
```

“Field name” (or index) Format specification

```
'3.141592653589793115997963468544 can be rounded to 3.142'
```

In this example, `.30` means 30 digits total precision.



built-in function value to format format specification (no { } or :)

```
format(ord("\x4c"),"08b")
```

```
'01001100'
```

Left-pad with zeros Total width: 8 Binary

See `string` documentation for much more about format specifiers.