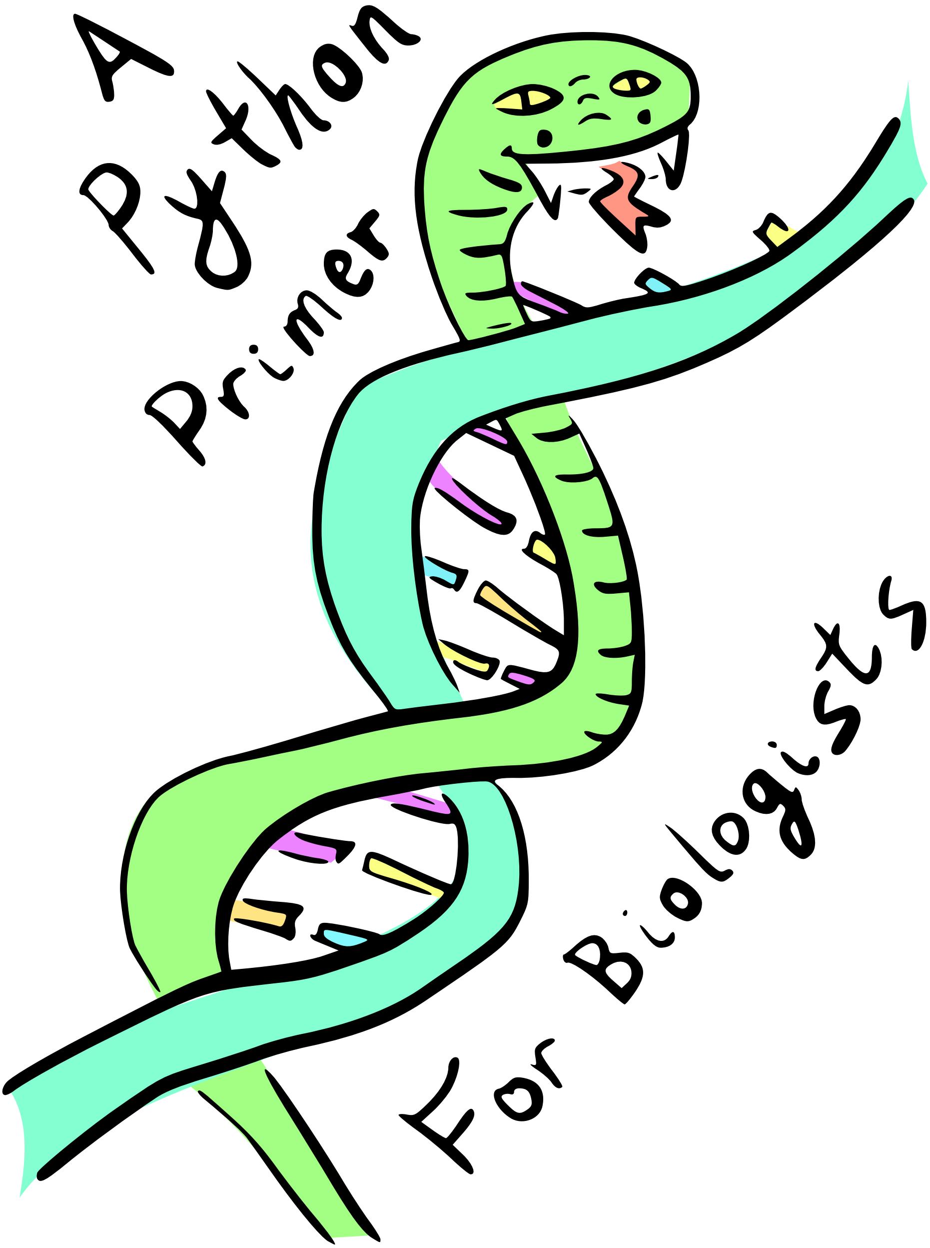
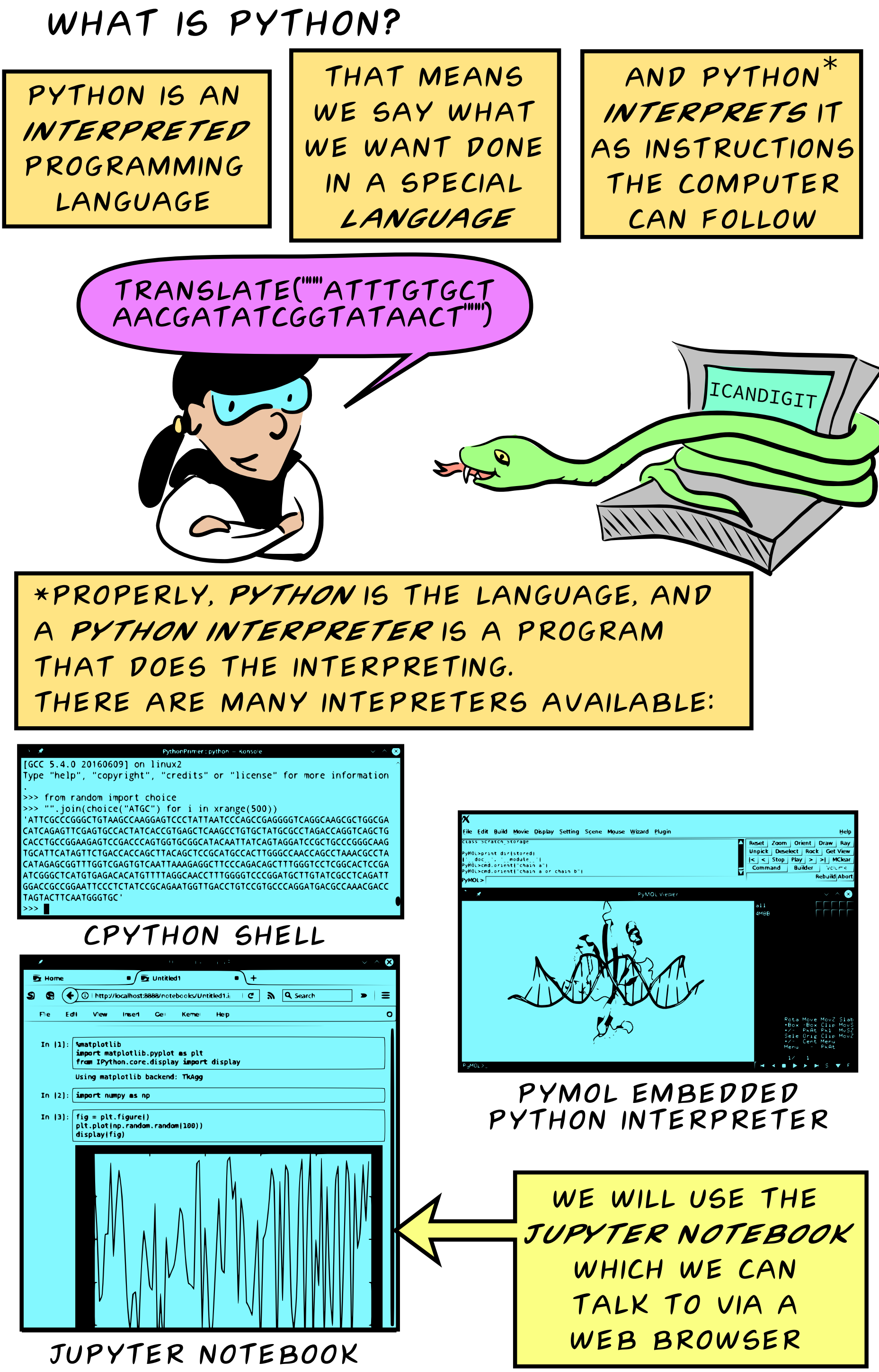

A Python Primer
For Biologists

Version 0.03
git commit
e2c05f3266

For screen and print pdfs of this primer, see

http://histo.ucsf.edu/BMS270/PythonPrimer.htm

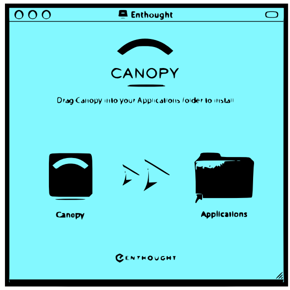

WHAT IS PYTHON?
PYTHON IS AN INTERPRETED PROGRAMMING LANGUAGE
THAT MEANS WE SAY WHAT WE WANT DONE IN A SPECIAL LANGUAGE
AND PYTHON* INTERPRETS IT AS INSTRUCTIONS THE COMPUTER CAN FOLLOW
TRANSLATE("""ATTTGTGCT AACGATATCGGTATAACT""")
ICANDIGIT
*PROPERLY, PYTHON IS THE LANGUAGE, AND A PYTHON INTERPRETER IS A PROGRAM THAT DOES THE INTERPRETING. THERE ARE MANY INTEPRETERS AVAILABLE:
CPYTHON SHELL
PYMOL EMBEDDED PYTHON INTERPRETER
WE WILL USE THE JUPYTER NOTEBOOK WHICH WE CAN TALK TO VIA A WEB BROWSER
JUPYTER NOTEBOOK

INSTALLING THE JUPYTER NOTEBOOK WITH ENTHOUGHT CANOPY EXPRESS

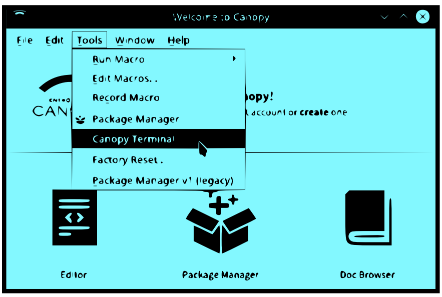
1) DOWNLOAD THE CANOPY EXPRESS INSTALLER FOR YOUR PLATFORM FROM:

HTTPS://STORE.ENTHOUGHT.COM/DOWNLOADS/#DEFAULT

2) FOLLOW THE PLATFORM SPECIFIC INSTRUCTIONS TO INSTALL CANOPY EXPRESS, CHOOSING THE PYTHON 3.5 INSTALLER



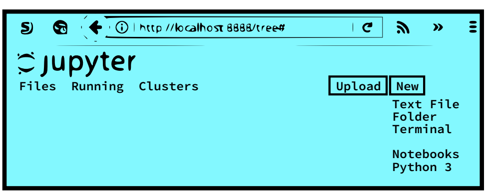
3) FROM CANOPY, CHOOSE TOOLS: CANOPY TERMINAL



4) IN THE TERMINAL, TYPE: `jupyter notebook`

THIS WILL OPEN THE JUPYTER NOTEBOOK SERVER IN YOUR DEFAULT WEB BROWSER

5) FROM YOUR WEB BROWSER, CHOOSE PYTHON3 FROM THE "NEW" MENU



THIS WILL OPEN A PYTHON NOTEBOOK IN A NEW BROWSER TAB OR WINDOW

6) YOU'RE READY TO CODE! IN YOUR NEW NOTEBOOK CELL TYPE "HELLO, WORLD" AND PRESS SHIFT-RETURN (MAC) OR SHIFT-ENTER (PC)

IN THIS GUIDE, WE WILL SHOW *INPUT CELLS* AS ROUND BOXES

```
[ ] "Hello, world"
```

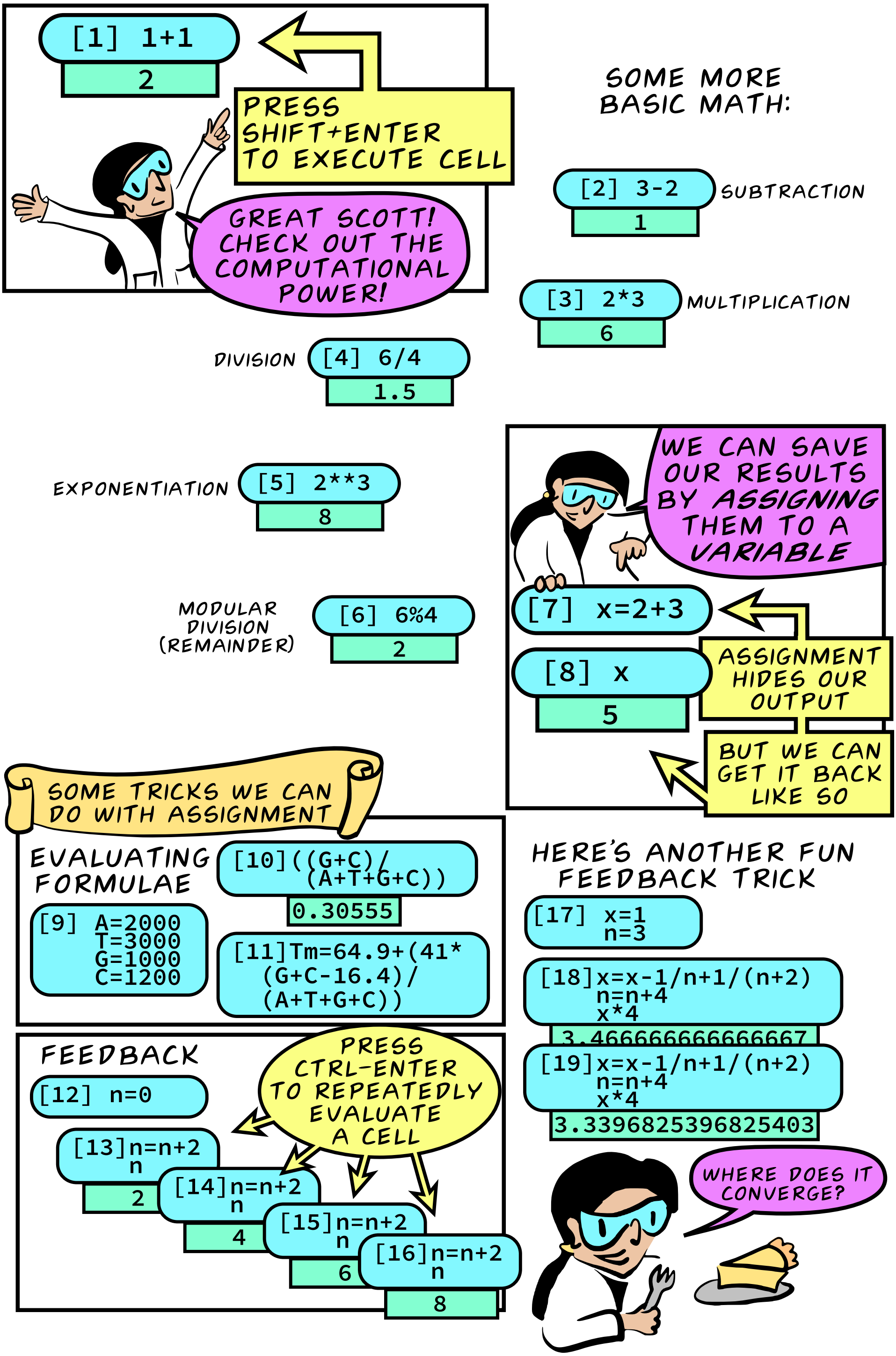
```
Hello, world
```

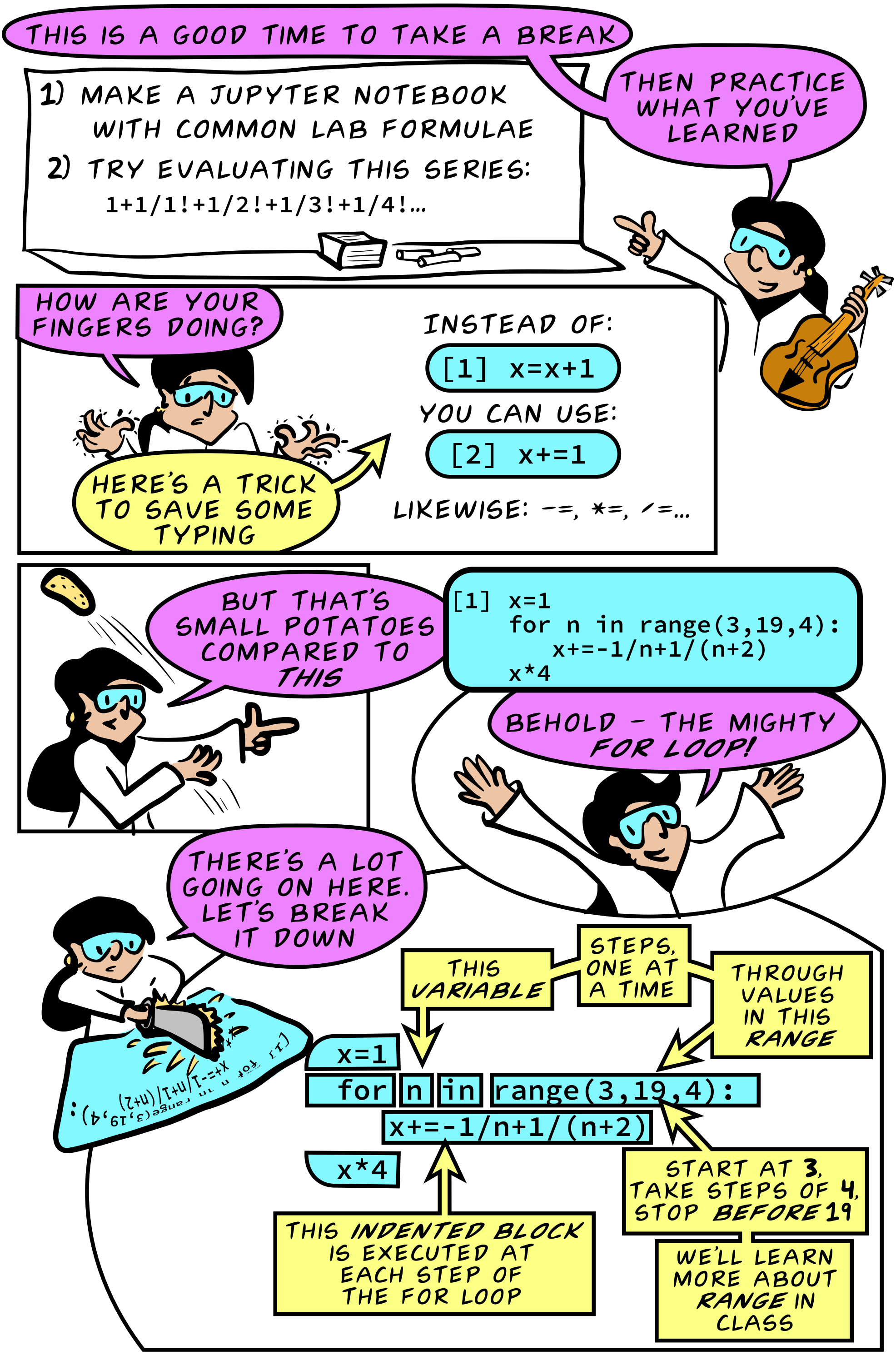
AND *OUTPUT CELLS* AS RECTANGLES

WHEN PYTHON EXECUTES YOUR INPUT, IT WILL ADD A NUMBER IN THE BRACKETS NEXT TO THE INPUT CELL

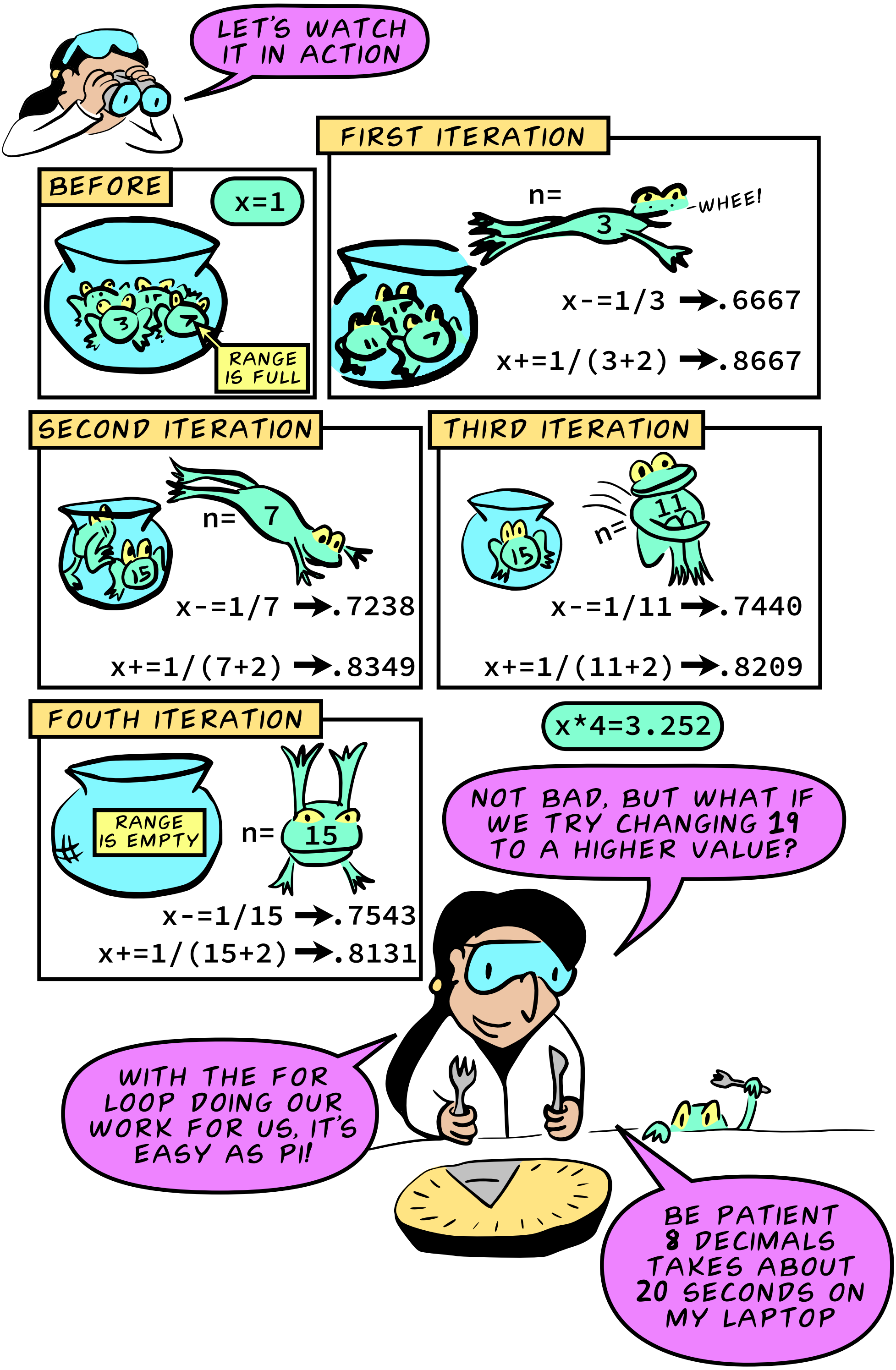
7) NOW MOVE ON TO THE NEXT PAGE AND FOLLOW ALONG IN YOUR NOTEBOOK!

YOUR FIRST NOTEBOOK: PYTHON AS A CALCULATOR

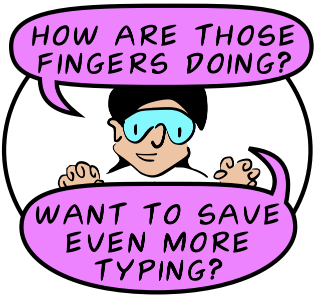




THIS IS A GOOD TIME TO TAKE A BREAK
1) MAKE A JUPYTER NOTEBOOK WITH COMMON LAB FORMULAE
2) TRY EVALUATING THIS SERIES:
1+1/1!+1/2!+1/3!+1/4!...
THEN PRACTICE WHAT YOU'VE LEARNED
HOW ARE YOUR FINGERS DOING?
HERE'S A TRICK TO SAVE SOME TYPING
INSTEAD OF:
[1] x=x+1
YOU CAN USE:
[2] x+=1
LIKEWISE: -=, *=, /=...
BUT THAT'S SMALL POTATOES COMPARED TO THIS
[1] x=1
for n in range(3,19,4):
x+=-1/n+1/(n+2)
x*4
BEHOLD - THE MIGHTY FOR LOOP!
THERE'S A LOT GOING ON HERE. LET'S BREAK IT DOWN
THIS VARIABLE
STEPS, ONE AT A TIME
THROUGH VALUES IN THIS RANGE
x=1
for n in range(3,19,4):
x+=-1/n+1/(n+2)
x*4
START AT 3, TAKE STEPS OF 4, STOP BEFORE 19
THIS INDENTED BLOCK IS EXECUTED AT EACH STEP OF THE FOR LOOP
WE'LL LEARN MORE ABOUT RANGE IN CLASS



LET'S WATCH IT IN ACTION
BEFORE
x=1
RANGE IS FULL
FIRST ITERATION
n=
3
WHEE!
x-=1/3 ➔.6667
x+=1/(3+2) ➔.8667
SECOND ITERATION
n=
7
x-=1/7 ➔.7238
x+=1/(7+2) ➔.8349
THIRD ITERATION
n=
11
x-=1/11 ➔.7440
x+=1/(11+2) ➔.8209
FOUTH ITERATION
RANGE IS EMPTY
n=
15
x-=1/15 ➔.7543
x+=1/(15+2) ➔.8131
x*4=3.252
NOT BAD, BUT WHAT IF WE TRY CHANGING 19 TO A HIGHER VALUE?
WITH THE FOR LOOP DOING OUR WORK FOR US, IT'S EASY AS PI!
BE PATIENT 8 DECIMALS TAKES ABOUT 20 SECONDS ON MY LAPTOP



JUST AS WE CAN SAVE *DATA* IN A VARIABLE WE CAN SAVE CODE IN A *FUNCTION*

DEFINE A FUNCTION NAMED PI WITH ONE PARAMETER (N)

WE USE THE *PARAMETER* (N) TO SET THE NUMBER OF TERMS THAT WE CALCULATE

AS IN A FOR LOOP THE *BODY* OF THE FUNCTION GOES IN AN *INDENTED BLOCK*

```
[1] def pi(N):
      x=1
      for n in range(3,3+4*N,4):
        x+=-1/n+1/(n+2)
      return x*4
```

WHEN WE *CALL* A FUNCTION WE INCLUDE PARAMETER VALUES IN *PARENTHESIS*

THE *RETURN* STATEMENT IS RESPONSIBLE FOR RETURNING THE RESULT OF THE FUNCTION

```
[2] pi(10)
3.1891847822776
```

```
[3] pi(1000000)
3.1415931535896
```

FUNCTIONS CAN TAKE MORE THAN ONE PARAMETER

```
[4] def add(x,y):
      return x+y
```

```
[5] add(3,5)
8
```

NO PARAMETERS

```
[6] def fortytwo():
      return 42
```

```
[7] fortytwo()
42
```

OR *OPTIONAL* PARAMETERS

```
[8] def f(x=3,y=5):
      return x-y
```

```
[9] f()
-2
```

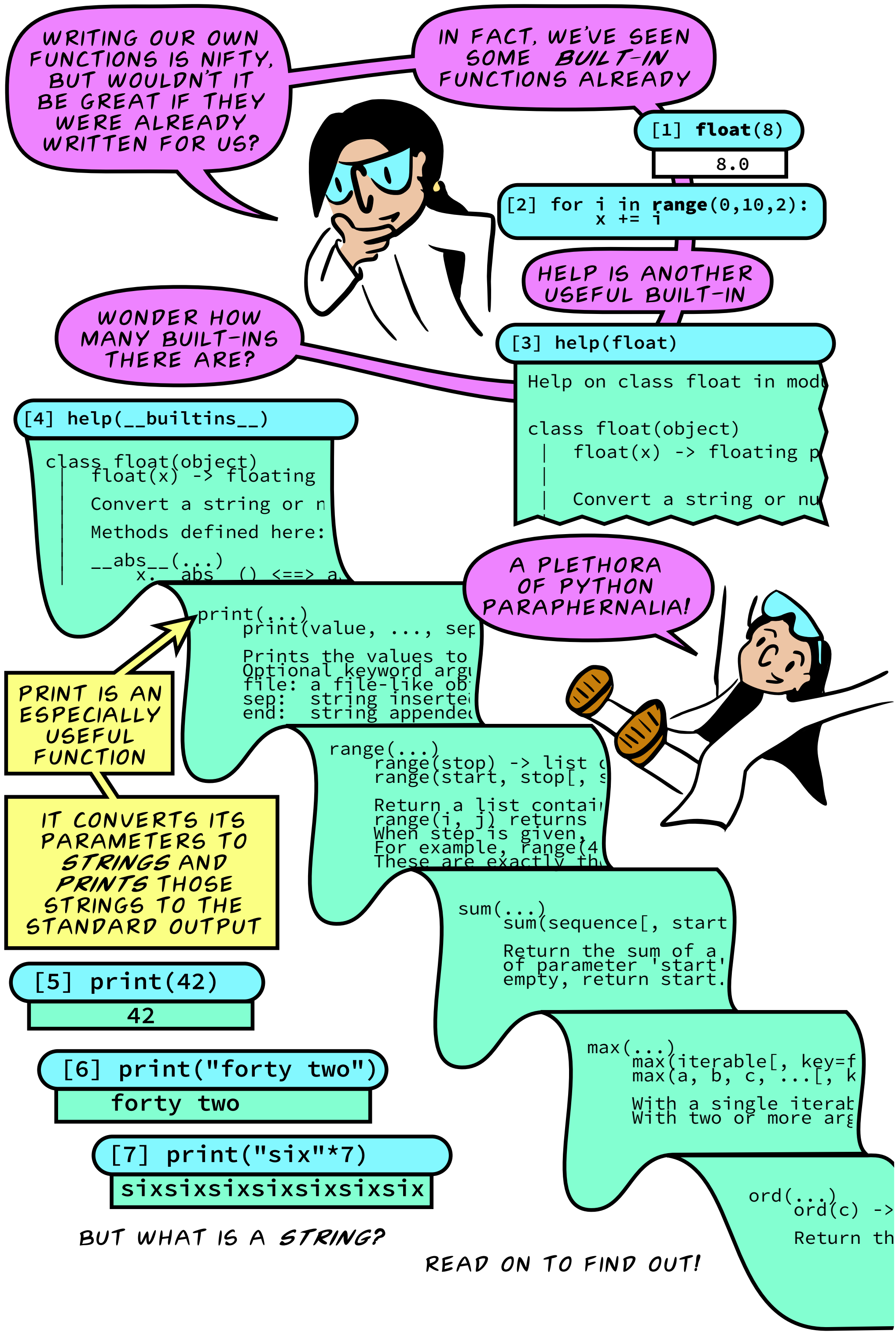
BY DEFAULT, PARAMETERS GET PLUGGED IN LEFT TO RIGHT

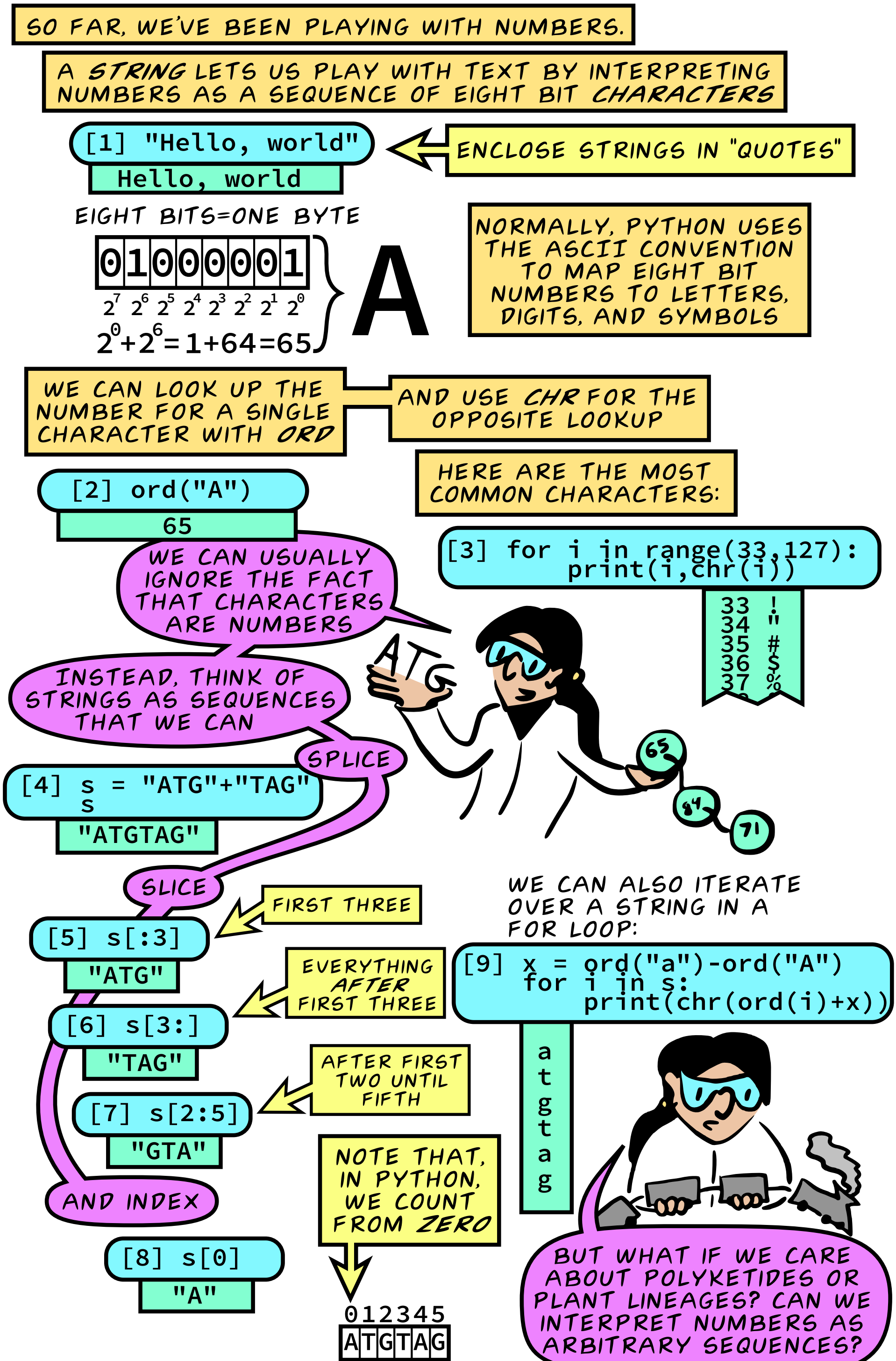
```
[10] f(11)
6
```

YOU CAN OVERRIDE THE DEFAULT BY *NAMING* A PARAMETER

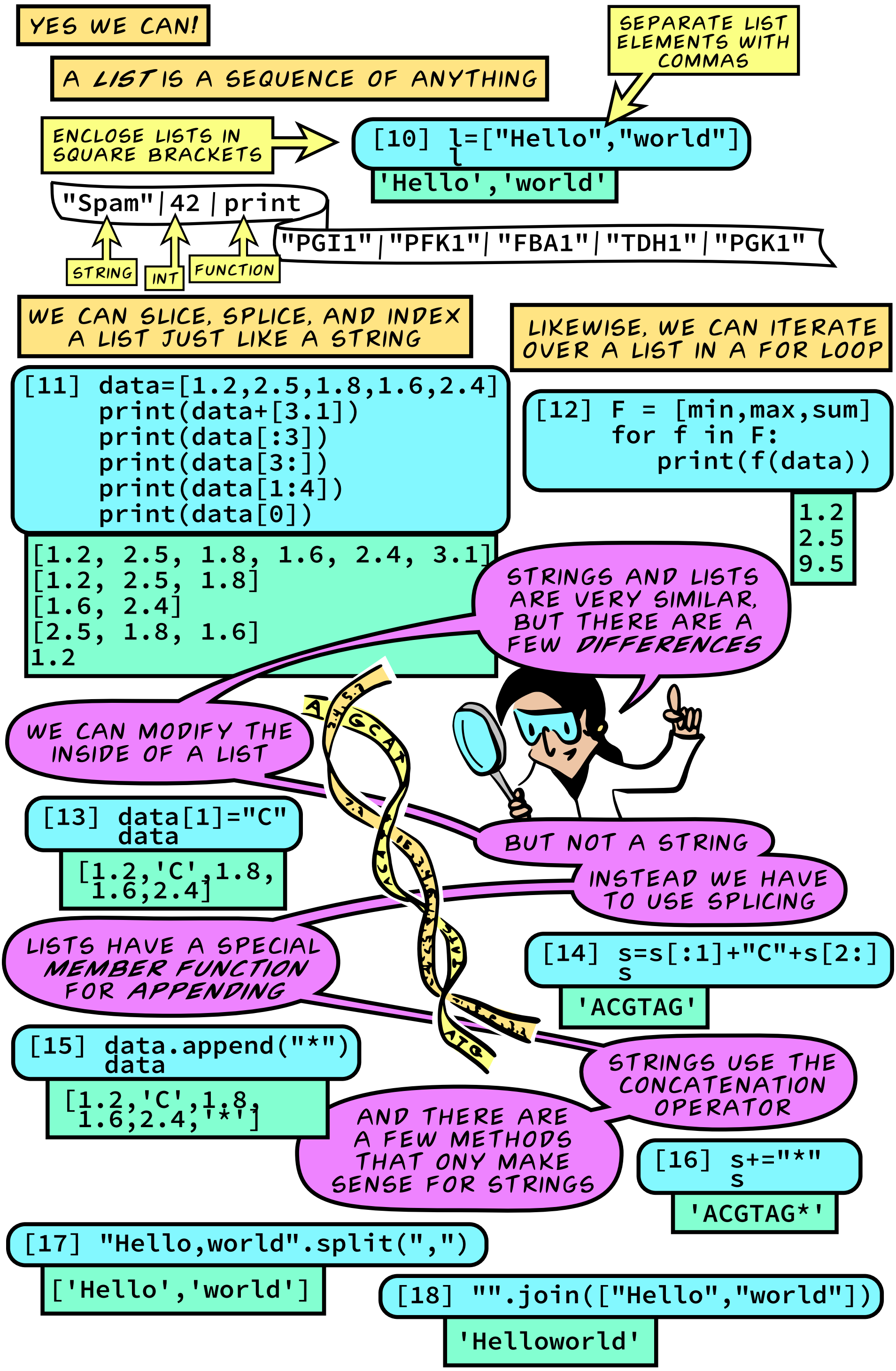
```
[11] f(y=1)
2
```

```
[12] f(8,7)
1
```


WRITING OUR OWN FUNCTIONS IS NIFTY, BUT WOULDN'T IT BE GREAT IF THEY WERE ALREADY WRITTEN FOR US?
IN FACT, WE'VE SEEN SOME *BUILT-IN* FUNCTIONS ALREADY
[1] float(8)
8.0
[2] for i in range(0,10,2):
x += i
HELP IS ANOTHER USEFUL BUILT-IN
WONDER HOW MANY BUILT-INS THERE ARE?
[3] help(float)
Help on class float in modu
class float(object)
| float(x) -> floating p
|
| Convert a string or nu
[4] help(__builtins__)
class float(object)
| float(x) -> floating
| Convert a string or n
| Methods defined here:
| __abs__(...)
| x.__abs__() <==> a
A PLETHORA OF PYTHON PARAPHERNALIA!
print(...)
print(value, ..., sep
Prints the values to
Optional keyword argu
file: a file-like obj
sep: string inserted
end: string appended
PRINT IS AN ESPECIALLY USEFUL FUNCTION
IT CONVERTS ITS PARAMETERS TO *STRINGS* AND *PRINTS* THOSE STRINGS TO THE STANDARD OUTPUT
range(...)
range(stop) -> list o
range(start, stop[, s
Return a list contai
range(i, j) returns
When step is given,
For example, range(4
These are exactly th
sum(...)
sum(sequence[, start
Return the sum of a
of parameter 'start'
empty, return start.
[5] print(42)
42
[6] print("forty two")
forty two
[7] print("six"*7)
sixsixsixsixsixsixsix
max(...)
max(iterable[, key=f
max(a, b, c, ...[, k
With a single iterab
With two or more arg
ord(...)
ord(c) ->
Return th
BUT WHAT IS A *STRING*?
READ ON TO FIND OUT!



SO FAR, WE'VE BEEN PLAYING WITH NUMBERS.
A STRING LETS US PLAY WITH TEXT BY INTERPRETING NUMBERS AS A SEQUENCE OF EIGHT BIT CHARACTERS
[1] "Hello, world"
Hello, world
ENCLOSE STRINGS IN "QUOTES"
EIGHT BITS=ONE BYTE
01000001
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
2^0+2^6=1+64=65
A
NORMALLY, PYTHON USES THE ASCII CONVENTION TO MAP EIGHT BIT NUMBERS TO LETTERS, DIGITS, AND SYMBOLS
WE CAN LOOK UP THE NUMBER FOR A SINGLE CHARACTER WITH ORD
AND USE CHR FOR THE OPPOSITE LOOKUP
[2] ord("A")
65
HERE ARE THE MOST COMMON CHARACTERS:
[3] for i in range(33,127):
print(i,chr(i))
33 !
34 "
35 #
36 $
37 %
WE CAN USUALLY IGNORE THE FACT THAT CHARACTERS ARE NUMBERS
ATG
INSTEAD, THINK OF STRINGS AS SEQUENCES THAT WE CAN
SPLICE
65
84
71
[4] s = "ATG"+"TAG"
s
"ATGTAG"
SLICE
FIRST THREE
[5] s[:3]
"ATG"
EVERYTHING AFTER FIRST THREE
[6] s[3:]
"TAG"
AFTER FIRST TWO UNTIL FIFTH
[7] s[2:5]
"GTA"
AND INDEX
[8] s[0]
"A"
NOTE THAT, IN PYTHON, WE COUNT FROM ZERO
012345
ATGTAG
WE CAN ALSO ITERATE OVER A STRING IN A FOR LOOP:
[9] x = ord("a")-ord("A")
for i in s:
print(chr(ord(i)+x))
a
t
g
t
a
g
BUT WHAT IF WE CARE ABOUT POLYKETIDES OR PLANT LINEAGES? CAN WE INTERPRET NUMBERS AS ARBITRARY SEQUENCES?


YES WE CAN!
A LIST IS A SEQUENCE OF ANYTHING
SEPARATE LIST ELEMENTS WITH COMMAS
ENCLOSE LISTS IN SQUARE BRACKETS
[10] l=["Hello","world"]
l
'Hello','world'
"Spam" | 42 | print
STRING
INT
FUNCTION
"PGI1" | "PFK1" | "FBA1" | "TDH1" | "PGK1"
WE CAN SLICE, SPLICE, AND INDEX A LIST JUST LIKE A STRING
[11] data=[1.2,2.5,1.8,1.6,2.4]
print(data+[3.1])
print(data[:3])
print(data[3:])
print(data[1:4])
print(data[0])
[1.2, 2.5, 1.8, 1.6, 2.4, 3.1]
[1.2, 2.5, 1.8]
[1.6, 2.4]
[2.5, 1.8, 1.6]
1.2
LIKEWISE, WE CAN ITERATE OVER A LIST IN A FOR LOOP
[12] F = [min,max,sum]
for f in F:
print(f(data))
1.2
2.5
9.5
STRINGS AND LISTS ARE VERY SIMILAR, BUT THERE ARE A FEW DIFFERENCES
WE CAN MODIFY THE INSIDE OF A LIST
[13] data[1]="C"
data
[1.2,'C',1.8,
1.6,2.4]
BUT NOT A STRING
INSTEAD WE HAVE TO USE SPLICING
[14] s=s[:1]+"C"+s[2:]
s
'ACGTAG'
LISTS HAVE A SPECIAL MEMBER FUNCTION FOR APPENDING
[15] data.append("*")
data
[1.2,'C',1.8,
1.6,2.4,'*']
STRINGS USE THE CONCATENATION OPERATOR
[16] s+="*"
s
'ACGTAG*'
AND THERE ARE A FEW METHODS THAT ONY MAKE SENSE FOR STRINGS
[17] "Hello,world".split(",")
['Hello','world']
[18] "".join(["Hello","world"])
'Helloworld'

THAT WAS A LOT TO DIGEST. TAKE SOME TIME TO PRACTICE FUNCTIONS, STRINGS, AND LISTS.

AS A FINAL EXERCISE, TRY USING SPLIT, JOIN, CHR, AND ORD TO WRITE ENCRYPTION AND DECRYPTION FUNCTIONS FOR A CIPHER LIKE THIS:

```
[1] encrypt("APPLE")
'BQQMF'
[2] encrypt("BQQMF")
'APPLE'
```

I HOPE THIS PRIMER WAS USEFUL. IF YOU'RE FEELING LOST, TRY WORKING SLOWLY THROUGH ALL OF THE EXAMPLES, AND TRY DOING THE EXERCISES.

(AND WE'LL COVER ALL OF THIS IN CLASS)

FOR MORE ON GETTING STARTED WITH PYTHON, I HIGHLY RECOMMEND

MARK LUTZ'S "LEARNING PYTHON"
http://search.safaribooksonline.com/9781449355722

AND MARK PILGRIM'S "DIVE INTO PYTHON 3"
http://histo.ucsf.edu/BMS270/diveintopython3-r802.pdf

SEE YOU IN CLASS!
http://histo.ucsf.edu/BMS270/