Level 1

Birds & Coconuts

Numbers & Variables

TRY PYTHON

Why Python?

Easy to read **Fast** **Versatile**

Random color art
jeremykun.com

Code School User Voice

Frets on Fire
fretsonfire.sourceforge.net

The Python Interpreter

>>> This symbol means "write your code here"

This symbol points to the outcome

```
>>> 30.5
```

```
30.5
```

Mathematical Operators

addition

```
>>> 3 + 5
```

→ 8

subtraction

```
>>> 10 - 5
```

→ 5

multiplication

```
>>> 3 * 5
```

→ 15

division

```
>>> 30 / 6
```

→ 5

exponent

```
>>> 2 ** 3
```

→ 8

negation

```
>>> -2 + -3
```

→ -5

Integers and Floats

Here's a little Python terminology.

int

```
>>> 35
```

```
35
```

An int is a
whole number

float

```
>>> 30.5
```

```
30.5
```

A float is a
decimal number

Order of Operations

These are the calculations happening first.

```
>>> ( 5 + 7 ) * 3
```

Behind the scenes

```
12 * 3
```

```
36
```

```
>>> 5 + 7 * 3
```

```
5 + 21
```

```
26
```

PEMDAS: **P**arentheses **E**xponent **M**ultiplication **D**ivision **A**ddition **S**ubtraction

Can a Swallow Carry a Coconut?

Let's use some math to figure out if this swallow can carry this coconut!

What we know: A swallow weighs 60g and can carry 1/3 of its weight.

Swallow's weight in grams

Divide by 3

```
>>> 60 / 3
```

```
20
```

So, 1 swallow can carry **20** grams

...but a coconut weighs **1450** grams

TRY PYTHON

How Many Swallows Can Carry a Coconut?

Let's try to figure out how many swallows it takes to carry 1 coconut.

What we know: A swallow weighs 60g and can carry 1/3 of its weight.

Coconut weight Swallow's weight Divide by 3

```
>>> 1450 / 60 / 3
```

```
8.06
```

That number seems way too low.
Let's look at what happened behind the scenes:

```
1450 / 60 / 3
```

```
24.17 / 3
```

We wanted 60/3 to happen first — we can add parentheses to fix this.

TRY PYTHON

Using Correct Order of Operations

How many swallows does it take to carry 1 coconut?

What we know: A swallow weighs 60g and can carry 1/3 of its weight.

Coconut weight

Swallow's weight

Divide by 3

```
>>> 1450 / ( 60 / 3 )
1450 / ( 20 )
```

```
72.5
```

That's a lot of swallows!

TRY PYTHON

What Other Fruits Can a Swallow Carry?

Let's see if we can get closer to 1 swallow with other fruits.

swallows per coconut:

```
>>> 1450 / ( 60 / 3 )
```

→ 72.5

swallows per apple:

```
>>> 128 / ( 60 / 3 )
```

→ 6.4

swallows per cherry:

```
>>> 8 / ( 60 / 3 )
```

→ 0.4

Great!
We found one that works!

Seems like we're repeating this a lot

Creating a Variable in Python

Use variables to store a value that can be used later in your program.

```
>>> swallow_limit = 60 / 3
```

variable name

value to store

swallows per coconut:

```
>>> 1450 / swallow_limit
```

→ 72.5

swallows per apple:

```
>>> 128 / swallow_limit
```

→ 6.4

swallows per cherry:

```
>>> 8 / swallow_limit
```

→ 0.4

- ✅ Less repeated calculations
- ✅ More organized

TRY PYTHON

Using Variables Assigns Meaning

Variables keep your code more readable and assign meaning to values.

```
>>> swallow_limit = 60 / 3
>>> swallows_per_cherry = 8 / swallow_limit
>>> swallows_per_cherry
```

```
0.4
```

What Should I Name My Variables?

As long as you're not breaking these rules, you can name variables anything you want!

```
no spaces = 0
```

no spaces in the name

```
3mice, $mice
```

no digits or special characters in front

Pep 8 Style Guide recommends:

```
swallow_limit
```

lowercase, use underscores for spaces

Pep 8 Style Guide does NOT recommend:

```
swallowLimit
```

camel case, later words are capitalized

Check Pep 8 style guide here - http://go.codeschool.com/pep8-styleguide

Can a Macaw Carry a Coconut?

Step 1: Declare variables for each value and find out the macaw's carrying limit.

```
>>> perc = 1/3
>>> coco_wt = 1450
>>> macaw_wt = 900
>>> macaw_limit = macaw_wt * perc
>>> macaw_limit
```

→ `300`

Notice our variables are descriptive, but we're still using abbreviations where appropriate.

TRY PYTHON

Can a Macaw Carry a Coconut?

Step 2: Divide the coconut's weight by the limit.

macaws needed to carry a coconut

```
>>> perc = 1/3
>>> coco_wt = 1450
>>> macaw_wt = 900
>>> macaw_limit = macaw_wt * perc
>>> num_macaws = coco_wt/macaw_limit
>>> num_macaws
```

```
4.8
```

TRY PYTHON

Importing Modules — Extra Functionality

Occasionally, we will want to use ***modules*** containing functionality that is not built in to the Python language.

This is importing extra math functions

```
>>> import math
>>> num_macaws = 4.8
>>> math.ceil(num_macaws)
```

ceil() is short for ceiling, which rounds up

5

Check out Python libraries here - http://go.codeschool.com/python-libraries

Can a Macaw Carry a Coconut?

Step 3: Use the ceiling function to round up.

```
>>> import math
>>> perc = 1/3
>>> coco_wt = 1450
>>> macaw_wt = 900
>>> macaw_limit = macaw_wt * perc
>>> num_macaws = coco_wt/macaw_limit
>>> math.ceil(num_macaws)
```

→ 5

TRY PYTHON

Level 2

Spam & Strings

Strings

Creating Strings

Strings are a sequence of characters that can store letters, numbers, or a combination — anything!

Create a string with quotes

```
string >>> 'Hello World'
```

```
'Hello World'
```

Single or double quotes work — just be consistent

```
string >>> "SPAM83"
```

```
'SPAM83'
```

String Concatenation With +

We can combine (or concatenate) strings by using the + operator.

```
>>> first_name = 'Monty'
>>> last_name = 'Python'
>>> full_name = first_name + last_name
>>> full_name
```

Need to add a space!

```
'MontyPython'
```

Concatenating a Space

```
>>> first_name = 'Monty'
>>> last_name = 'Python'
>>> full_name = first_name + ' ' + last_name
>>> full_name
```

We can concatenate a space character between the words

```
'Monty Python'
```

Moving Our Code to a Script File

```
>>> first_name = 'Monty'
>>> last_name = 'Python'
>>> full_name = first_name + ' ' + last_name
>>> full_name
```

Each line is entered on the command line

Instead, we can put all lines into a single script file

script.py

```
first_name = 'Monty'
last_name = 'Python'
full_name = first_name + ' ' + last_name
```

But how can we output the value of full_name?

Output From Scripts With print()

Everything in 1 script:

script.py

```
first_name = 'Monty'
last_name = 'Python'
full_name = first_name + ' ' + last_name
print(full_name)
```

Prints whatever is inside the ()

```
Monty Python
```

```
print(first_name, last_name)
```

```
Monty Python
```

print() as many things as you want, just separate them with commas

print() automatically adds spaces between arguments

In Python 2, print doesn't need ()

```
print full_name
```

Running a Python Script

script.py

```
first_name = 'Monty'
last_name = 'Python'
full_name = first_name + ' ' + last_name
print(full_name)
```

Put the name of your script file here

```
>>> python script.py
```

This is the command to run Python scripts

```
Monty Python
```

Demo: Running a Python Script From the Console

MontyPython.py - /Users/sbuchanan/Desktop/MontyPython.py (3.4.3)

```
first_name = 'Monty'
last_name = 'Python'
full_name = first_name + ' ' + last_name
print(full_name)
```

Ln: 5 Col: 0

Desktop — bash — 76×17

```
SarahBs-MBP:Desktop sbuchanan$
```

Comments Describe What We're Doing

Let's write a script to describe what Monty Python is.

script.py

```
# Describe the sketch comedy group
name = 'Monty Python'
description = 'sketch comedy group'
year = 1969
```

means this line is a comment and doesn't get run

Concatenating Strings and Ints

When we try to concatenate an int, year, with a string, we get an error.

script.py

```
# Describe the sketch comedy group
name = 'Monty Python'
description = 'sketch comedy group'
year = 1969

# Introduce them

sentence = name + ' is a ' + description + ' formed in ' + year
```

Year is an int, not a string

```
TypeError: Can't convert 'int' object to str implicitly
```

TRY PYTHON

Concatenating Strings and Ints

script.py

```
# Describe the sketch comedy group
name = 'Monty Python'
description = 'sketch comedy group'
year = '1969'

# Introduce them

sentence = name + ' is a ' + description + ' formed in ' + year
```

We could add quotes and make the year a string instead of an int

This will work, but it really makes sense to keep numbers as numbers.

Turning an Int Into a String

script.py

```
# Describe the sketch comedy group
name = 'Monty Python'
description = 'sketch comedy group'
year =  1969

# Introduce them
sentence = name + ' is a ' + description + ' formed in ' + str(year)
print(sentence)
```

Instead, convert the int to a string when we concatenate with str()

```
Monty Python is a sketch comedy group formed in 1969
```

print() Casts to String Automatically

script.py

```
# Describe the sketch comedy group
name = 'Monty Python'
description = 'sketch comedy group'
year = 1969

# Introduce them
print(name, 'is a', description, 'formed in', year)
```

print() does string conversions for us

```
Monty Python is a sketch comedy group formed in 1969
```

Dealing With Quotes in Strings

script.py

```
# Describe Monty Python's work
famous_sketch = 'Hell's Grannies'
```

Interpreter thinks the quote has ended and doesn't know what S is

```
Syntax Error: invalid syntax
```

script.py

```
# Describe Monty Python's work
famous_sketch1 = "Hell's Grannies"

print(famous_sketch1)
```

Start with " " instead — now the " no longer means the end of the string

```
Hell's Grannies
```

TRY PYTHON

Special Characters for Formatting

We want to add some more sketches and print them out

Let's add another famous sketch

script.py

```
# Describe Monty Python's work
famous_sketch1 = "Hell's Grannies"
famous_sketch2 = 'The Dead Parrot'

print(famous_sketch1, famous_sketch2)
```

```
Hell's Grannies The Dead Parrot
```

This works, but we want to format it better.

Special Characters — Newline

\n is a special character that means new line.

script.py

```
# Describe Monty Python's work
famous_sketch1 = "\nHell's Grannies"
famous_sketch2 = '\nThe Dead Parrot'

print('Famous Work:', famous_sketch1, famous_sketch2)
```

Add a newline
to make this look better

```
Famous Work:
Hell's Grannies
The Dead Parrot
```

This works, but we want to indent the titles.

TRY PYTHON

Special Characters — Tab

\t is a special character that means tab.

script.py

```
# Describe Monty Python's work
famous_sketch1 = "\n\tHell's Grannies"
famous_sketch2 = '\n\tThe Dead Parrot'

print('Famous Work:', famous_sketch1, famous_sketch2)
```

Add a tab to indent and make this look even better

```
Famous Work:
    Hell's Grannies
    The Dead Parrot
```

Strings Behind the Scenes — a List of Characters

A string is a list of characters, and each character in this list has a position or an index.

```
greeting = 'H E L L O  W O R L D !'
```

[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10][11]

Starts at 0

```
>>> greeting[0]
```

→ `'H'`

```
>>> greeting[11]
```

→ `'!'`

We can get each character by calling its index with []

```
>>> greeting[12]
```

→ `IndexError: string index out of range`

String Built-in Function — len()

There are a few built-in string functions — like len(), which returns the length of a string and is very useful.

```
'H E L L O  W O R L D !'
```

[0] [1] [2] [3] [4] [5] [6] [7] [8][9][10][11]

script.py

```
# print the length of greeting
greeting = 'HELLO WORLD!'
print( len(greeting) )
```

12

```
12
```

The Man Who Only Says Ends of Words

'Good'	'Evening'
[0] [1] [2] [3]	[0] [1] [2] [3] [4] [5] [6]

script.py

```
word1 = 'Good'
end1 = word1[2] + word1[3]
print(end1)
```

The last half of the word: characters at positions [2] and [3]

→ od

TRY PYTHON

Using Slices to Access Parts of a String

Boundary starts before index 2

Boundary ends before index 5

```
word =  "P y t h o n"
```

[0] [1] [2] [3] [4] [5]

```
>>> word[2:5]
```

→ `tho`

Start boundary

End boundary

Slice Formula and Shortcuts

slice formula: variable [start : end+1]

```
word1 = G o o d'
```

[0] [1] [2] [3]

```
word1 ='G o o d
```

[0] [1] [2] [3]

```
>>> word1[0:2]
```

→ G o

```
>>> word1[2:4]
```

→ o d

Shortcuts:

Means from start to position

```
>>> word1[:2]
```

→ G o

Means from position to end

```
>>> word1[2:]
```

→ o d

Incorporating String Slices Into Our Solution

'Good'	'Evening'
[0] [1] [2] [3]	[0] [1] [2] [3] [4] [5] [6]

script.py

```
word1 = 'Good'
end1 = ~~word1[2] + word1[3]~~ word1[2:4]
print(end1)
```

Replace this with a slice

→ od

TRY PYTHON

Using the String Slice Shortcut

'Good'	'Evening'
[0] [1] [2] [3]	[0] [1] [2] [3] [4] [5] [6]

script.py

```
word1 = 'Good'
end1 = word1[2:]

word2 = 'Evening'
end2 = word2[3:]

print(end1, end2)
```

Improved this with a shortcut

→ `od ning`

It would be great if we didn't have to know the halfway points were at 2 and 3...

The Index at the Halfway Point

The len() function will let us calculate the halfway index of our word.

script.py

```
# Calculate the halfway index
word1 = 'Good'
half1 = len(word1)/2

word2 = 'Evening'
half2 = len(word2)/2
```

half1 is 2.0

half2 is 3.5

PROBLEM: indexes have to be integers — floats won't work!

'G o o d'

[0] [1] [2] [3]

'E v e n i n g'

[0] [1] [2] [3] [4] [5] [6]

The Index at the Halfway Point

Integer division vs floor division. Be consistent if we cover in Level 1.

// 2 division signs means integer division in Python.

script.py

```
# Calculate the halfway index
word1 = 'Good'
half1 = len(word1)//2


word2 = 'Evening'
half2 = len(word2)//2
```

half1 is 4//2 = 2

// Means integer division

half2 is 7//2 = 3

// Also rounds down to the nearest integer

'Good'

[0] [1] [2] [3]

'Evening'

[0] [1] [2] [3] [4] [5] [6]

In Python 2, single division / is int division unless there are floats involved

Making Our Program More Reusable

Calculating the halfway index makes our logic reusable.

half1 is 4//2 = 2

half2 is 7//2 = 3

script.py

```
word1 = 'Good'
half1 = len(word1)//2
end1 = ~~word1[2:]~~   word1[half1:]

word2 = 'Evening'
half2 = len(word2)//2
end2 = ~~word2[3:]~~   word2[half2:]

print(end1, end2)
```

```
od ning
```

'Good'

[0] [1] [2] [3]

'Evening'

[0] [1] [2] [3] [4] [5] [6]

Level 3

Conditional Rules of Engagement

Conditionals & Input

TRY PYTHON

Extended Battle Rules

```
if there's less than 3 knights
Retreat!
```

```
if there's more than 10 knights
Trojan Rabbit!
```

```
otherwise
```

Comparison Operators

There are **6** comparators in Python:

<	<=	==	>=	>	!=
less than	less than equal to	equal to	greater than equal to	greater than	not equal to

Is 5 less than 10 ??

```
>>> 5 < 10
```

→ `True`

Is 5 equal to 10 ??

```
>>> 5 == 10
```

→ `False`

Is name equal to Jim ??

```
>>> name = 'Pam'
>>> name == 'Jim'
```

Setting the name variable equal to 'Pam'

→ `False`

Is name NOT equal to Jim ??

```
>>> name != 'Jim'
```

→ `True`

TRY PYTHON

Comparison Operator Outputs

Here's the output of evaluating all 6 comparisons:

```
>>> 5 < 10
True

>>> 5 == 10
False

>>> 5 > 10
False

>>> 5 <= 10
True

>>> 5 != 10
True

>>> 5 >= 10
False
```

Booleans

In programming, a boolean is a type that can only be True or False.

True

booleans

False

Capital T and Capital F

```
>>>  is_raining = True
>>>  is_raining
```

True

Conditional — if, then

An if statement lets us ***decide*** what to do: if ***True***, then ***do*** this.

script.py

```
# Battle Rules
num_knights = 2

if num_knights < 3:
    print('Retreat!')
    print('Raise the white flag!')
```

True

If True:
Then do this
Then this

Any indented code that comes after an if statement is called a code block

```
Retreat!
Raise the white flag!
```

TRY PYTHON

Conditional — if, then

An if statement lets us ***decide*** what to do: if ***False***, then ***don't*** do this.

script.py

```
# Battle Rules
num_knights = 4

if num_knights < 3:
    print('Retreat!')
    print('Raise the white flag!')
```

False

Because the conditional is False, this block doesn't run

Nothing happens...
crickets

The Program Continues After the Conditional

The code continues to run as usual, line by line, after the conditional block.

script.py

```
# Battle Rules
num_knights = 4

if num_knights < 3:
    print('Retreat!')
    print('Raise the white flag!')
print('Knights of the Round Table!')
```

False

Always this

This block doesn't run, but the code right outside of the block still does

```
Knights of the Round Table!
```

TRY PYTHON

Rules for Whitespace in Python

In Python, indent with 4 spaces. However, anything will work as long as you're consistent.

Irregular spacing is a no-no!

2 spaces

4 spaces

```
if num_knights == 1:
    # Block start
  print('Do this')
    print('And this')
print('Always this')
```

```
IndentationError: unexpected indent
```

The PEP 8 style guide recommends 4 spaces - http://go.codeschool.com/pep8-styleguide

Extended Battle Rules

```
if there's less than 3 knights
Retreat!
```

```
otherwise
```

Conditional — if False → else

script.py

```
# Battle Rules
num_knights = 4

if num_knights < 3:
    print('Retreat!')
else:
    print('Truce?')
```

If this statement is **True**,
then run the indented code below

Otherwise,
then run the indented code below

→ `Truce?`

TRY PYTHON

Conditionals — How to Check Another Condition?

script.py

```
# Battle Rules
num_knights = 10

if num_knights < 3:
    print('Retreat!')
else:
    print('Truce?')
```

Truce?

Otherwise if at least 10, then Trojan Rabbit

TRY PYTHON

elif Allows Checking Alternatives

Else sequences will exit as soon as a statement is True...

script.py

```
# Battle Rules
num_knights = 10
day = 'Wednesday'

if num_knights < 3:
    print('Retreat!')
elif num_knights >= 10:
    print('Trojan Rabbit')
elif day == 'Tuesday':
    print('Taco Night')
else:
    print('Truce?')
```

elif means otherwise if

We can have multiple elifs

...and continue the program after

```
Trojan Rabbit
```

TRY PYTHON

Combining Conditionals With and/or

King Arthur has decided:

On all **Mondays**, we retreat

```
if num_knights < 3:
    print('Retreat!')
```

OR

if it's a Monday

Retreat!

We can only use the Trojan Rabbit on **Wednesday**
(we're renting it out on other days)

```
if num_knights >= 10:
    print('Trojan Rabbit!')
```

AND

it's Wednesday

Trojan Rabbit!

Putting or in Our Program

On all **Mondays**, or if we have less than 3 knights, we retreat.

If we have less than 3 knights OR If it's a Monday

```
if num_knights < 3 or day == "Monday":
    print('Retreat!')
```

Spelled out Monday, and Wednesday on the next slide.

Putting and in Our Program

We can only use the Trojan Rabbit if we have at least 10 knights AND it's Wednesday.

If we have at least 10 knights AND If it's a Wednesday

```
if num_knights >= 10 and day == "Wednesday":
    print('Trojan Rabbit!')
```

Boolean Operators — and / or

Make it possible to combine comparisons or booleans.

If 1 is False
and result will be False

True	*and*	True	→	True
True	*and*	False	→	False
False	*and*	False	→	False

If 1 is True
or result will be True

True	*or*	True	→	True
True	*or*	False	→	True
False	*or*	False	→	False

Incorporating the Days of the Week

script.py

```
# Battle Rules
num_knights = 10
day = 'Wednesday'

if num_knights < 3 or day == 'Monday':
    print('Retreat!')
elif num_knights >= 10 and day == 'Wednesday':
    print('Trojan Rabbit!')
else:
    print('Truce?')
```

Checking day of the week

User Input — With the input() Function

script.py

```
# Ask the user to input the day of the week
day = input('Enter the day of the week')

print('You entered:', day)
```

day saves the user's input

prints out this statement and waits for user input

Desktop — bash — 73×13

```
SarahBs-MacBook-Pro:Desktop sbuchanan$
```

In Python 2, raw_input() is used instead of input()

Receiving User Input From Console

Just like we can print() text to the console, we can also get user input from the console.

script.py

```
# Ask the user to input the number of knights
num_knights = int(input('Enter the number of knights'))

print('You entered:', num_knights)
if num_knights < 3 or day == 'Monday':
    print('Retreat!')
```

Change (or cast) the string to an int

User enters #, but it comes in as text

```
Enter the number of knights 10
```

User enters 10

```
You entered: 10
```

Different Input and Conditionals Changes Output

Different user input causes different program flow:

1st run

2nd run

script.py

```
# Battle Rules
num_knights = int(input('How many knights?'))
day = input('What day is it?'))

if num_knights < 3 or day == 'Monday':
    print('Retreat!')
elif num_knights >= 10 and day == 'Wednesday':
    print('Trojan Rabbit!')
else:
    print('Truce?')
```

1st run

How many knights?	2
What day is it?	Tuesday
Retreat	

2nd run

How many knights?	12
What day is it?	Wednesday
Trojan Rabbit!	

Extended Battle Rules

killer bunny

Holy
Hand Grenade

num_knights < 3

num knights >= 10

otherwise

Nested Conditionals — Ask Follow-up Questions

script.py

```
# Battle Rules
num_knights = int(input('Enter number of knights')
day = input('Enter day of the week')
enemy = input('Enter type of enemy')

if enemy == 'killer bunny':
    print('Holy Hand Grenade!')
else:
```

First, find out if our enemy is the killer bunny

Do the original battle rules here

```
# Original Battle Rules
if num_knights < 3 or day == 'Monday':
    print('Retreat!')
if num_knights >= 10 and day == 'Wednesday':
    print('Trojan Rabbit!')
else:
    print('Truce?')
```

If not, then use the original battle rules

Nested Conditionals

script.py

```
# Battle Rules
num_knights = int(input('Enter number of knights')
day = input('Enter day of the week')
enemy = input('Enter type of enemy')

if enemy == 'killer bunny':
    print('Holy Hand Grenade!')
else:
    # Original Battle Rules
    if num_knights < 3 or day == 'Monday':
        print('Retreat!')
    if num_knights >= 10 and day == 'Wednesday':
        print('Trojan Rabbit!')
    else:
        print('Truce?')
```

Our final
Rules of Engagement

TRY PYTHON