Why Python

- Popular
- Easy to learn and use
- Open-source
- General-Purpose
- Multi-paradigm (procedureal, object-oriented, functional)
- Hettinger, “What makes Python Awesome?”[**?**]
  `http://tinyurl.com/mndd4er`
- Did I mention *popular*?

Conceptual Hierarchy by Mark Lutz[**?**]

- Programs are composed of *modules*.
- Modules contain *statements*.

Conceptual Hierarchy by Mark Lutz[**?**]

- Programs are composed of *modules*.
- Modules contain *statements*.
- Statements contain expressions.
- Expressions create and process objects.

Script File (.py)

- A script file is a module.
- A script is a sequence of statements, delimited by a newline (or the end of line character).
- Python executes one statement at a time, from the top of a script file to the bottom.

Script File (.py)

- A script file is a module.
- A script is a sequence of statements, delimited by a newline (or the end of line character).
- Python executes one statement at a time, from the top of a script file to the bottom.
- Execution happens in namespaces (modules, classes, functions all have their own).
- Everything is runtime (even `def`, `class`, and `import`)

Executable Python Script File

- On a Unix-like system, we can change the mode of the script file and make it executable:

```
$chmod +x hello.py
$./hello.py
```

- Just add the first line with a hashbang (#!):

```
#!/usr/bin/python

# say hello to the world
def main():
    print "Hello, World!"

if __name__ == "__main__":
    main()
```

Comments

- Comments start with a hash (#) and end with a newline.

```
1  # this whole line is a comment
2
3  # add 10 integers.
4  total = 0
5  for i in range(10): # i goes 0, 1, 2, ..., 9
6      total += i      # shortcut for total = total + i
7
8  print "total=", total
9  # total= 45
```

Variables

- Variables are created when first assigned a value.

```
my_var = 3
answer_to_everything = 42

# also works are:
x = y = 0
a, b, c = 1, 2, 3
```

- Variable names start with a letter or an underscore(_) and can have letters, underscores, or digits.
- Variable names are case-sensitive.

Assignment Semantics According to David Godger[?]

- Variables in many *other languages* are a container that stores a value.

```
1  int a = 1;
```

Assignment Semantics According to David Godger[?]

- Variables in many *other languages* are a container that stores a value.

```
1  int a = 1;
```

- In Python, an assignment creates an object, and labels it with the variable.

```
1  a = 1
```

Assignment Semantics According to David Godger[?]

- Variables in many *other languages* are a container that stores a value.

```
1  int a = 1;
```

- In Python, an assignment creates an object, and labels it with the variable.

```
1  a = 1
```

- If you assign another value to `a`, then the variable labels the new value (`2`).

```
1  a = 2
```

Assignment Semantics According to David Godger[?]

- Variables in many *other languages* are a container that stores a value.

```
1  int a = 1;
```

- In Python, an assignment creates an object, and labels it with the variable.

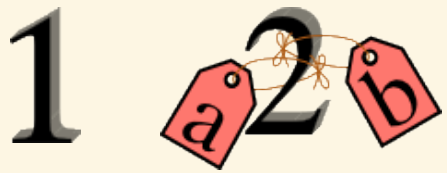
```
1  a = 1
```

- If you assign another value to `a`, then the variable labels the new value (`2`).

```
1  a = 2
```

- This is what happens if you assign `a` to a new variable `b`:

```
1  b = a
```



Numbers

▶ Integers

```
x = 0
age = 20
size_of_household = 5

print type(age)
# <type 'int'>

# can handle arbitrarily large numbers
huge = 10 ** 100 + 1
```

▶ Floating-point Numbers

```
g = 0.1
f = 6.67384

velocity = 1.   # it is the dot (.) that makes it a float
print velocity
# 1.0
type(velocity)
# <type 'float'>
```

Numeric Expressions

- Most of the arithmetic operators behave as expected.

```
a = 10
b = 20
print a - (b ** 2) + 23
# -367

x = 2.0
print x / 0.1
# 20.0
```

- Watch out for integer divisions. In Python 2, it *truncates down* to an integer.

```
print 10 / 3
# 3 (in Python 2)          3.3333333333333335 (in Python 3)
print -10 / 3
# -4 (in Python 2)          -3.3333333333333335 (in Python 3)

# a solution: use floating point numbers
print 10.0 / 3.0
# 3.3333333333333335 (in both Python 2 and 3)
```

Quiz

- What is the remainder of 5 divided by 2?
- An answer:

```
# an answer
dividend = 5
divisor = 2

quotient = dividend // divisor
remainder = dividend - (quotient * divisor)
print "remainder:", remainder

# another answer
print "remainder:", dividend  divisor
```

- What is the remainder of 2837465 divided by 2834?
- An answer using the modulus operator (%):

```
print 2837465 % 2834
```

String Literals

- A string is a sequence of characters.

```
1  # either double (") or single(') quotes for creating string literals.
2  name = "Changarilla"
3  file_name = 'workshop.tex'
4
5  # triple-quoted string
6  starwars = """
7  A long time ago is a galaxy far, far away...
8
9  It is a period of civil war. Rebel
10 spaceships, striking from a hidden
11 base, have won their first victory
12 ...
13
14 What is the last character of this string?
15 """
16
17 last_char = starwars[-1]
18 print ord(last_char), ord("\n")
19 # 10 10
```

Working with strings

- Strings are immutable.

```
s = "abcde"
s[0] = "x" # trying to change the first char to "x"
# TypeError: 'str' object does not support item assignment

t = "x" + s[1:] # creating a new string
print t
# xbcde
```

- Many functions and methods are available.

```
s = "abcde"
print s + s # concatenation
# abcdeabcde

print len(s)
# 5

print s.find("c") # index is 0-based. returns -1 if not found
# 2
```

String Manipulation

- A few more string methods.

```
1  s = "abcde"
2  print s.upper(), "XYZ".lower()
3  # ABCDE, xyz
4
5  print "   xxx  yy  ".strip()
6  # xxx yy
7
8  print "a,bb,ccc".split(",")
9  # ['a', 'bb', 'ccc']
```

String Manipulation

- A few more string methods.

```
1  s = "abcde"
2  print s.upper(), "XYZ".lower()
3  # ABCDE, xyz
4
5  print "   xxx  yy  ".strip()
6  # xxx yy
7
8  print "a,bb,ccc".split(",")
9  # ['a', 'bb', 'ccc']
```

- What are "methods"?

String Manipulation

- A few more string methods.

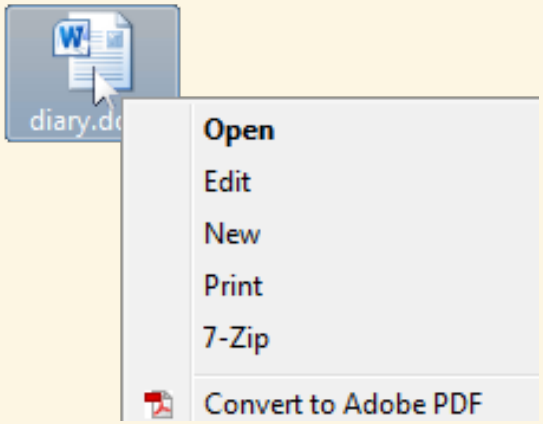
```
1  s = "abcde"
2  print s.upper(), "XYZ".lower()
3  # ABCDE, xyz
4
5  print "   xxx  yy  ".strip()
6  # xxx yy
7
8  print "a,bb,ccc".split(",")
9  # ['a', 'bb', 'ccc']
```

- What are "methods"?
  - Functions which are a member of a type (or class).

String Manipulation

- A few more string methods.

```
1  s = "abcde"
2  print s.upper(), "XYZ".lower()
3  # ABCDE, xyz
4
5  print "   xxx  yy  ".strip()
6  # xxx yy
7
8  print "a,bb,ccc".split(",")
9  # ['a', 'bb', 'ccc']
```



- What are "methods"?
  - Functions which are a member of a type (or class).
  - int, str, Word Document are types (or classes).
  - 2, "abcde", diary.docx are an *instance* (or *object*) of the respective type.
  - Types have members: properties (data) and methods (functions).

Two Ways to Format

- `format()` method.

```
print "The answer is {0}".format(21)
# The answer is 21
print "The real answer is {0:6.4f}".format(21.2345678)
# The answer is 21.2346
```

- Formatting operator.

```
print "The answer is %d" % 21
# The answer is 21
print "The real answer is %6.4f" % 21.2345678
# The answer is 21.2346
```

Quiz

- Say hello to Simba and friends. Use print statement(s). The output should look like below. (Hint: Use {0:s} or %s for the place holder.)

```
Hello, Simba!
Hello, Timon!
Hello, Pumbaa!
```

- An answer:

```
friends = ["Simba", "Timon", "Pumbaa"]
hello_template = "Hello, %s!"

for friend in friends:
    print hello_template % friend
```

Raw Strings

- Within a string literal, escape sequences start with a backslash (\)

```
a_string = 'It\'s a great day\nto learn \\Python\\.\n 1\t 2\t 3'
print a_string
# It's a great day
# to learn \Python\.
# 1       2       3
```

- A *raw* string literal starts with the prefix `r`. In a raw string, the backslash (\) is not special.

```
import re

# raw strings are great for writing regular expression patterns.
p = re.compile(r"\d\d\d-\d\d\d\d")
m = p.match('123-4567')
if m is not None:
    print m.group()
# 123-4567
```

Unicode Strings

- You can create Unicode strings using the u prefix and "\u" escape sequences.

```
1  a_string = u"Euro \u20AC" # \u followed by 16-bit hex value xxxx
2  print a_string, len(a_string)
3  # Euro € 6
```

Unicode Strings

- You can create Unicode strings using the `u` prefix and "\u" escape sequences.

```
1  a_string = u"Euro \u20AC" # \u followed by 16-bit hex value xxxx
2  print a_string, len(a_string)
3  # Euro € 6
```

- Unicode strings are sequences of *code points*.
- Code points are numbers, each representing a “character”. e.g., U+0061 is ‘Latin small letter a’.
- Unicode text strings are *encoded* into bytes. UTF-8 is one of many Unicode encodings, using one to four bytes to store a Unicode “character”.
- Once you have Unicode strings in Python, all the string functions and properties work as expected.

Best Practices According to Thomas Wouters[?]

- Never mix unicode and bytecode [i.e. ordinary] strings.
- Decode bytecode strings on input.
- Encode unicode strings on output.
- Try automatic conversion (codecs.open())
- Pay attention to exceptions, UnicodeDecodeError
- An example

```
1  ustr = u"Euro \u20AC"   # Euro €
2
3  # Python's default encoding codec is 'ascii'
4  ustr.encode()
5  # UnicodeEncodeError: 'ascii' codec can't encode characater 'u\u20ac'
6  utf_8 = ustr.encode("UTF-8") # encoding to UTF-8 works fine
7  # this takes 8 bytes. five one-byte's and one three-byte
8
9  # now we want to decode to ascii, ignoring non-ascii chars
10 print utf_8.decode("ascii", "ignore")
11 # Euro (no euro symbol character)
```

None

- Is a place-holder, like NULL in other languages.

```
x = None
```

- Is a universal object, i.e., there is only one None.

```
print None is None
# True
```

- Is evaluated as False.

```
x = None
if x:
    print "this will never print"
```

- Is, however, distinct from others which are False.

```
print None is 0, None is False, None is []
# False, False, False
```

Core Data Types

- Basic core data types: int, float, and str.
- "Python is dynamically, but *strongly* typed."

```
n = 1.0
print n + "99"
# TypeError: unsupported operand types(s) for +: 'int' and 'str'
```

- Use `int()` or `float()` to go from str to numeric

```
print n + float("99")
# 100.0
```

- `str()` returns the string representation of the given object.

```
n = 1.0
print str(n) + "99"
# 1.099
```

Operators

- Python supports following types of operators:
  - Arithmetic (+ − * / % ** //)
  - Comparison (== != > < >= <=)
  - Assignment (= += −= *= /= %= **= //=)
  - Logical (and or not)
  - Bitwise (& | ^ ~ << >>)
  - Membership (in not in)
  - Identity (is is not)

A Few Surprises

- Power operator (`**`) binds more tightly than unary operators on the left.

```
a = -2**2
print a
# -4
# solution: parenthesize the base
print (-2)**2
# 4
```

- Comparisons can be chained.

```
x < y <= z          # y is evaluated only once here
```

  is equivalent to

```
x < y and y <= z
```

- Logical operators (`and`, `or`) short-circuit evaluation and return an *operand*.

```
print 3 or 2
# 3
```

Quiz

- Demographic and Health Surveys (DHS) Century Month Code (CMC)[**?**, p.5] provides an easy way working with year and month.

  The CMC is an integer representing a month, taking the value of 1 in January 1900, 2 in February 1900, . . . , 13 in January 1901, etc. The CMC in February 2011 is 1334.

  What is the CMC for this month, i.e., January, 2014?

Quiz

- Demographic and Health Surveys (DHS) Century Month Code (CMC)[**?**, p.5] provides an easy way working with year and month.

  The CMC is an integer representing a month, taking the value of 1 in January 1900, 2 in February 1900, . . . , 13 in January 1901, etc. The CMC in February 2011 is 1334.

  What is the CMC for this month, i.e., January, 2014?
- An answer:

```
1 month = 5
2 year = 2015
3 cmc = 12 * (year - 1900) + month
4 print cmc for year(%d) month(%d) is: %d" % (year, month, cmc)
5 # cmc for year(2015) month(5) is: 1385
```

Quiz

- Demographic and Health Surveys (DHS) Century Month Code (CMC)[**?**, p.5] provides an easy way working with year and month.

  The CMC is an integer representing a month, taking the value of 1 in January 1900, 2 in February 1900, . . . , 13 in January 1901, etc. The CMC in February 2011 is 1334.

  What is the CMC for this month, i.e., January, 2014?

- An answer:

```
1 month = 5
2 year = 2015
3 cmc = 12 * (year - 1900) + month
4 print cmc for year(%d) month(%d) is: %d" % (year, month, cmc)
5 # cmc for year(2015) month(5) is: 1385
```

- What is the month (and year) of CMC 1000?

```
1 print "cmc(%d) is year(%d) and month(%d)" % (1000,
2     (1900 + (1000 // 12)), (1000 % 12))
```

Quiz

- According to U.S. National Debt Clock, the outstanding public debt, as of a day in 2012, was a big number:

```
1  debt = 17234623718339.96
```

  Count how many times the digit 3 appears in the number. (Hint: create a string variable and use the `count()` method of the string type.)

Quiz

- According to U.S. National Debt Clock, the outstanding public debt, as of a day in 2012, was a big number:

```
1  debt = 17234623718339.96
```

  Count how many times the digit 3 appears in the number. (Hint: create a string variable and use the `count()` method of the string type.)

- An answer:

```
1  sdebt = "17234623718339.96"
2  print sdebt.count("3")
3  # 4
```

Quiz

- According to U.S. National Debt Clock, the outstanding public debt, as of a day in 2012, was a big number:

```
1  debt = 17234623718339.96
```

  Count how many times the digit 3 appears in the number. (Hint: create a string variable and use the count() method of the string type.)

- An answer:

```
1  sdebt = "17234623718339.96"
2  print sdebt.count("3")
3  # 4
```

- (tricky) It feels rather silly to rewrite the value as a string. Can you think of a way to *convert* the number into a string?
- An answer:

```
1  print "{0:17.2f}".format(debt).count("3")
2  # 4
```

Flow Control

- Conditional Execution (`if`)
- Iteration
  - `while` loop
  - `for` loop

IF

- IF statement is used for conditional execution.

```
if x > 0:
    print "x is positive"
else:
    print "x is zero or negative"
```

- Only one suite (block of statements) under a True conditional expression is executed.

```
me = "rock"
wins = 0

if you == "paper":
    print "You win!"
elif you == "scissors":
    print "I win!"
    wins += 1
else:
    print "draw"
```

Compound statements

- `if`, `while`, and `for` are *compound* statements, which have one or more *clauses*. A *clause*, in turn, consists of a *header* that ends with a colon (`:`) and a *suite*.
- A suite, a block of statements, is identified by *indentation*.

```
1  a = 1
2  if a > 0:
3      desc = "a is positive"        # these two lines form a suite
4      print a, desc                 #
5                                    # (blank line ignored)
6  print "done"                      # This line being started "dedented"
7                                    # signals the end of the block
8
9      if a > 0:                     # indentation error
10 desc = "a is positive"            # indentation error
11
12 if a > 0:                         # OK
13         desc = "a is positive"    # OK
14         print a, desc             # OK
15 else:                             # OK
16     print a                       # OK
```

Compound statements

- The amount of indentation does not matter (as long as the same within a level). Four spaces (per level) and no tabs are the convention.
- Use editor's python mode, which prevents/converts <TAB> to (four) spaces.
- Why indentation? A good answer at: `http://tinyurl.com/kxv9vts`.
- For an empty suite, use `pass` statement, which does nothing.

Quiz

- Given an integer n, print out "Even" if n is an even number or "Odd" if n is an odd number. (Hint: (n % 2 == 0) is true when n is an even number.
- An answer:

```
n = 3
if (n % 2 == 0):
    print "Even"
else:
    print "Odd"
```

- Given an integer n, print out "Even" if n is an even number except zero or "Odd" if n is an odd number. When n is equal to zero (0), then print out "Even and zero", instead of just "Even".
- An answer:

```
n = 3  # using the if else expressions, not if else statements
print ("Odd" if (n% 2) else "Even") + ("" if n else " and zero")
```

WHILE

- Repeats a block of statements as long as the condition remains True.

```
1  total = 0
2  n = 0
3  while n < 10:
4      print n, total
5      n += 1
6      total += n
7  # 0 0
8  # 1 1
9  # 2 3
10 # ...
11 # 9 45
```

- break terminates the loop immediately.
- continue skips the remainder of the block and goes back to the test condition at the top.

FOR

- `for` is used to iterate over a sequence.

```
1  days = ["Sunday", "Monday", "Tuesday", "Wednesday",
2          "Thursday", "Friday", "Saturday"]
3
4  for day in days:
5      if day == "Friday":
6          print "I am outta here."
7          break
8      print "Happy" + " " + day + "!"
9
10 # Happy Sunday!
11 # Happy Monday!
12 # ...
13 # Happy Thusday!
14 # I am outta here.
```

FOR

- Another example.

```
1  numbers = range(5)
2  print numbers
3  # [0, 1, 2, 3, 4]
4
5  for n in numbers:
6      print n,
7      if n % 2 == 0:
8          print "is even"
9      else:
10         print "is odd"
11
12 # 0 is even
13 # 1 is odd
14 # 2 is even
15 # 3 is odd
16 # 4 is even
```

Quiz

- Write either a while or a for loop to add integers from 1 to a given positive integer, n (n >= 1). For example, when n is 3, your program should print out 6, when n is 10, 55. (Hint: It is easy to get an infinite loop. If you don't see output and kernel keeps running (indicated by the filled circle under the Python logo on the top right corner of ipython notebook), then interrupt the kernel by clicking on the ipython notebook menu, Kernel > Interrupt, and fix the error.)
- An answer:

```
1 n = 3
2 i = 1
3 total = 0
4 while (i <= n):
5     total = total + i
6     i = i + 1
7 print total
```

Quiz

- You may have noticed that it is rather silly to use a loop to calculate this sum. Calculate the sum of integers from 1 to n (n >= 1) directly without a loop. (Hint the sum is also known as the "triangular number".)
- An answer:

```
1  n = 3
2  print (n * (n + 1)) / 2
```

File I/O

- The built-in function, `open()`, returns a `file` type object, unless there is an error opening the file.

```
in_file = open("yourfile.txt", "r") # for reading
out_file = open("myfile.txt", "w") # for writing
```

- Once we get the `file` type object, then use its methods.

```
# read all the contents from the input file
content = in_file.read()

# write out a line to the output file
out_file.write("hello?\n")

# close it when done
in_file.close()
out_file.close()
```

Reading a file one line at a time

- with ensures that the file is closed when done.

```
1  with open("lorem.txt", "r") as f:
2      for line in f:
3          print line
```

- Another example.

```
1  # creating a file and writing three lines
2  with open("small.txt", "w") as f:
3      f.write("first\n")
4      f.write("second\n")
5      f.write("third")
6
7  with open("small.txt", "r") as f:
8      for line in f:
9          print line    # line includes the newline char
10 # first
11 #
12 # second
13 #
14 # third
```

Defining and Calling a Function

- Defined with `def` and called by name followed by `()`.

```
def the_answer():
    return 42

print the_answer()
# 42
```

- Argument(s) can be passed.

```
def shout(what):
    print what.upper() + "!"

shout("hello")
# HELLO!
```

- If the function returns nothing, then it returns `None`.

```
r = shout("hi")
# HI!
print r
# None
```

Another example

- CMC again

```
1  def cmc(year, month):
2      ''' returns DHS Century Month Code ''' # doc string
3      if year < 1900 or year > 2099:
4          print "year out of range"
5          return
6      if month < 1 or month > 12:
7          print "month out of range"
8          return
9      value = (year - 1900) * 12 + month
10     return value
11
12 print cmc(2014, 1)
13 # 1369
14 print cmc(2014, 15)
15 # month out of range
16 # None
```

Quiz

- Write a function odd(n) which returns true if given the number is odd or false otherwise. (Hint: Recall the remainder operator %.)

```
1  def odd(n):
2      return (n  2 == 0)
```

- Write a function, triangular(n), which returns the triangular number n, that is the sum of integers from 1 to the given number, n. For example, triangular(3) should return 6 and triangular(10) should return 55.

```
1  def triangular(n):
2      return (n * (n + 1)) / 2
```

- Print out the first 20 odd triangular numbers. (Hint: OEIS A014493)

```
1  for i in range(1, 21):
2      print triangular(i),
```

Local and Global Variables

- Within your function:
  - A new variable is *local*, and independent of the global var with the same name, if any.

```
1  x = 1                          # global (or module)
2
3  def my_func():
4      x = 2                      # local
5  my_func()
6  print x                        # 1
```

  - Both local and global variables can be read.
  - Global variables can be writen to once *declared* so.

```
1  x = 1
2
3  def my_func():
4      global x
5      x = 2                      # global
6  my_func()
7  print x                        # 2
```

Quiz

- Write a function that returns Body Mass Index (BMI) of an adult given weight in kilograms and height in meters.
  (Hint: BMI = weight(kg) / (height(m) squared).
  For instance, if a person is 70kg and 1.80m, then BMI is about 21.6.)

Quiz

- Write a function that returns Body Mass Index (BMI) of an adult given weight in kilograms and height in meters.
  (Hint: BMI = weight(kg) / (height(m) squared).
  For instance, if a person is 70kg and 1.80m, then BMI is about 21.6.)
- An answer.

```
1  def bmi(kg, m):
2      return float(kg) / (m ** 2)
```

Quiz

- Write a function that returns Body Mass Index (BMI) of an adult given weight in kilograms and height in meters. (Hint: BMI = weight(kg) / (height(m) squared).
  For instance, if a person is 70kg and 1.80m, then BMI is about 21.6.)
- An answer.

```
1  def bmi(kg, m):
2      return float(kg) / (m ** 2)
```

- Re-write the bmi function so that it accepts height in feet and inches, and the weight in pounds. (Hint. Make pound, foot, and inch arguments. Convert them into local variables, kg and m, before calculating bmi to return.)

```
1  def bmi(foot, inch, pound):
2      kg = 0.453592 * pound
3      m = 0.0254 * (12 * foot + inch)
4      return kg / (m ** 2)
```

Importing a module

- `import` reads in a module, runs it (top-to-bottom) to create the module object.
- Via the module object, you get access to its variables, functions, classes, . . .
- We've already seen an example of importing a standard regular expression module:

```
1  import re
2
3  # compile() is a function defined within the imported re module.
4
5  p = re.compile(r"\d\d\d-\d\d\d\d")
6  m = p.match('123-4567')
7   if m is not None:
8       print m.group()
9  # 123-4567
```

Another example

- There are many standard modules that come already installed, and be ready to be imported.

```
1  import math
2
3  s = math.sqrt(4.0)
4  print "4.0 squared is {0:.2f}".format(s)
5  # 4.0 squared is 2.00
```

- You can selectively import as well.

```
1  from math import sqrt # import sqrt() alone
2
3  print sqrt(9.0)       # no "math."
4  # 3.0
```

- You can import your own Python script file (.py) the same way. The default import path includes the current working directory and its sub-directories.

```
1  import hello    # suppose that hello.py defines a main() function
2
3  hello.main()
4  # "hello, World!"
```

Quiz

- Write a function such that, given a BMI value, returns the BMI category as a string. Recall the categories are:

Underweight	less than 18.5
Normal weight	18.5 upto but not including 25
Overweight	25 upto but not including 30
Obesity	30 or greater

For instance, the function should return a string "Normal weight", when it is called with an argument of, say 20. (Hint: use conditional statements i.e., `if` ... `elif` ...)

```
1  def bmi_category(bmi):
2      if bmi < 18.5:
3          return "Underweight"
4      elif bmi < 25.0:
5          return "Normal weight"
6      elif bmi < 30.0:
7          return "Overweight"
8      else:
9          return "Obesity"
```

Quiz

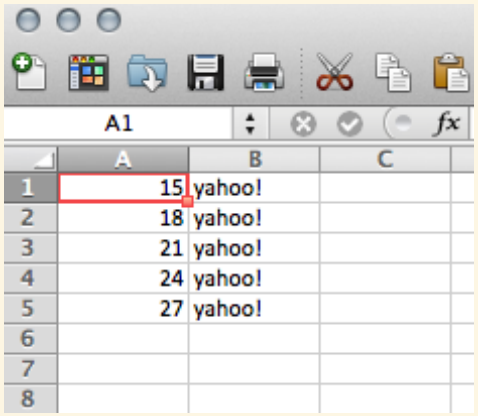
- Print out a BMI table showing several lines of a pair: a BMI value and its category. BMI value may start at 15 and go up by 3 up to 36. (Hint: use a loop)

```
1  for bmi in rnage(15, 36 + 1, 3):
2      print "\%3.0f \%s" \% (bmi, bmi_category(bmi))
```

Quiz (cont.)

- Create a comma-separated values (.csv) file of the BMI table. (Hint: You may start with below, and modify it as neccessary.)

```
1  import csv
2
3  with open("test.csv", "wb") as f:
4      my_writer = csv.writer(f)
5      bmi = 15
6      while bmi < 36:
7          cat = bmi_category(bmi)
8          my_writer.writerow([bmi, cat])
9          bmi += 3
```



Summary

- Using Python interactively or by running a script.
- Comments.
- Variables and assignment semantics.
- Core data types (int, float, str, None).
- Operators.
- Conditionals and Looping.
- Defining and using functions.
- Basic File I/O.
- Importing a module.

References