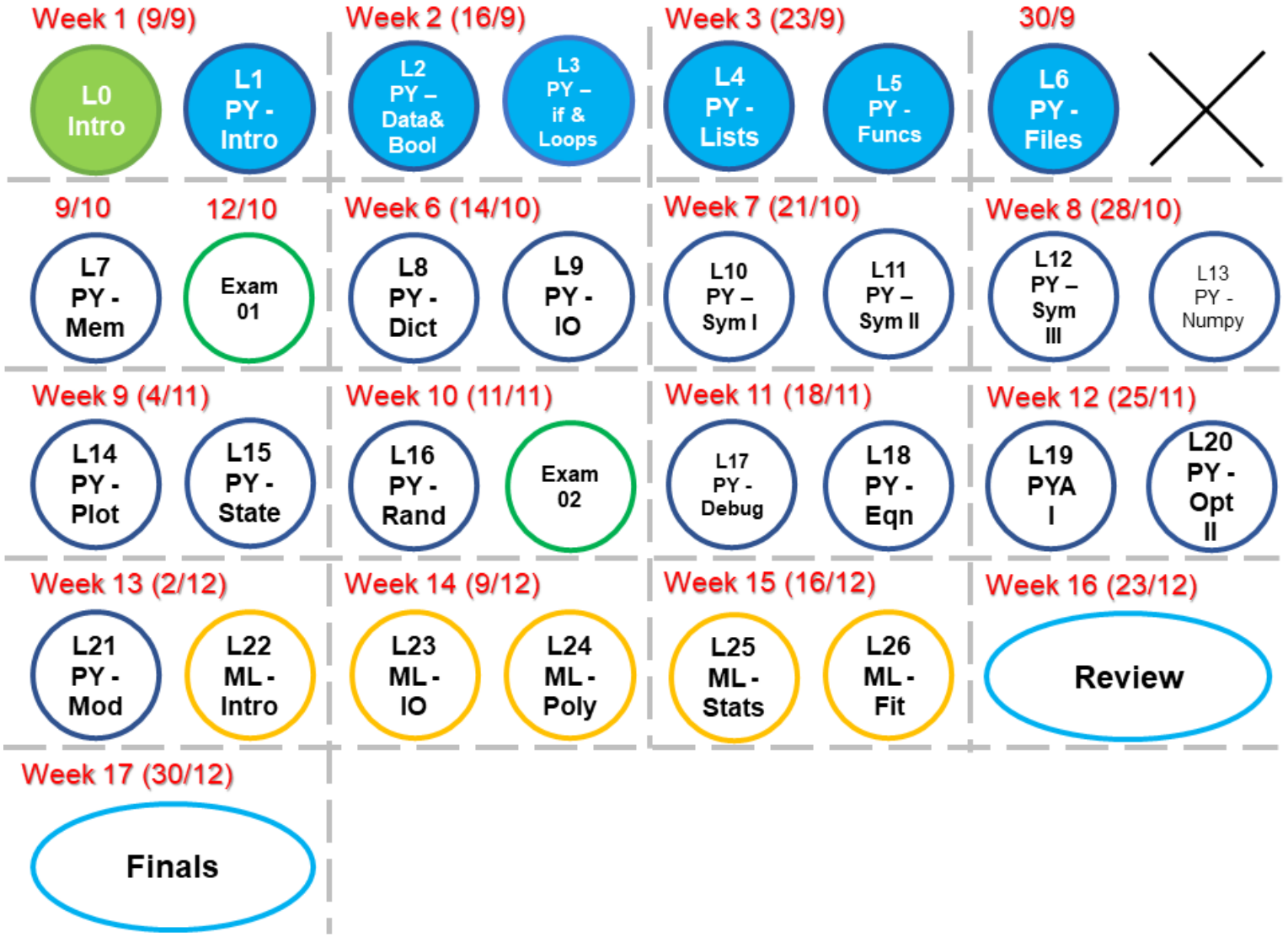
Python 101

File Operations

Roadmap


Week 1 (9/9)
L0 Intro
L1 PY - Intro
Week 2 (16/9)
L2 PY – Data& Bool
L3 PY – if & Loops
Week 3 (23/9)
L4 PY - Lists
L5 PY - Funcs
30/9
L6 PY - Files
9/10
L7 PY - Mem
12/10
Exam 01
Week 6 (14/10)
L8 PY - Dict
L9 PY - IO
Week 7 (21/10)
L10 PY – Sym I
L11 PY – Sym II
Week 8 (28/10)
L12 PY – Sym III
L13 PY - Numpy
Week 9 (4/11)
L14 PY - Plot
L15 PY - State
Week 10 (11/11)
L16 PY - Rand
Exam 02
Week 11 (18/11)
L17 PY - Debug
L18 PY - Eqn
Week 12 (25/11)
L19 PYA I
L20 PY - Opt II
Week 13 (2/12)
L21 PY - Mod
L22 ML - Intro
Week 14 (9/12)
L23 ML - IO
L24 ML - Poly
Week 15 (16/12)
L25 ML - Stats
L26 ML - Fit
Week 16 (23/12)
Review
Week 17 (30/12)
Finals

Announcements

quiz: `quiz06` due on Tues 10/01

lab: no `lab`

hw: `hw03` due TODAY (Mon 09/30)

exam: `exam01` 12 Oct `lec01-05` MCQ and short questions

Recap

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:
tt = 6

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

```
runningSum( 3 ):
```

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

```
runningSum( 3 ):
 if 3 == 0: #FALSE
```

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

```
runningSum( 3 ):
 if 3 == 0: #FALSE
 f = 3 + runningSum ( 2 ) #function waits: P1
```

...

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

...

```
runningSum( 2 ):
```

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

...

```
runningSum( 2 ):
 if 2 == 0: #FALSE
```

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

...

```
runningSum( 2 ):
 if 2 == 0: #FALSE
 f = 2 + runningSum ( 1 ) #function waits: P2
```

...

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

...

```
runningSum( 1 ):
 if 1 == 0: #FALSE
 f = 1 + runningSum ( 0 ) #function waits: P3
```

Recursive

Question:

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f

tt = runningSum( 3 )
```

answer:

...

```
runningSum( 1 ):
 if 1 == 0: #FALSE
 f = 1 + runningSum ( 0 ) #function waits: P3
runningSum( 0 )
 if 0 == 0: #TRUE
 return 0 #runningSum( 0 ) ends; returns to P3
```

...

Recursive

answer:

...

```
f = 1 + runningSum ( 0 ) #P3
f = 1 + 0 #P3
return 1 #runningSum ( 1 ) ends; return to P2
```

Recursive

answer:

...

```
f = 1 + runningSum ( 0 ) #P3
f = 1 + 0 #P3
return 1 #runningSum ( 1 ) ends; return to P2
f = 2 + runningSum ( 1 ) #P2
f = 2 + 1 #P2
return 3 #runningSum ( 2 ) ends; return to P1
```

Recursive

answer:

...

```
f = 1 + runningSum ( 0 ) #P3
f = 1 + 0 #P3
return 1 #runningSum ( 1 ) ends; return to P2
f = 2 + runningSum ( 1 ) #P2
f = 2 + 1 #P2
return 3 #runningSum ( 2 ) ends; return to P1
f = 3 + runningSum ( 2 ) #P1
f = 3 + 3 #P1
return 6 #runningSum ( 3 ) ends; return to tt
```

scope 1

Value of x at #1 and #2?

```
x = 4
x *= 2              # 1.

def do_calc(x):
    print(x)        # 2.
    return x ** 2
```

scope 1

Value of x at #1 and #2?

```
x = 4
x *= 2              # 1.

def do_calc(x):
    print(x)        # 2.
    return x ** 2
```

Ans:

1. 8

scope 1

Value of x at #1 and #2?

```
x = 4
x *= 2              # 1.

def do_calc(x):
    print(x)        # 2.
    return x ** 2
```

Ans:

1. 8
2. ?

scope 2

Value of x at #1 and #2?

```
x = 4
x *= 2              # 1.

def do_calc(x):
    print(x)        # 2.
    return x ** 2

do_calc(x)
```

Ans:
1. 8

scope 2

Value of x at #1 and #2?

```
x = 4
x *= 2              # 1.

def do_calc(x):
    print(x)        # 2.
    return x ** 2

do_calc(x)
```

Ans:
1. 8
2. 8

scope

Value of x at #1, #2 and #3?

```
x = 4
x *= 2              # 1.
def do_calc(x):
    print(x)        # 2.
    return x ** 2

do_calc(x)
y = x + 2
x = do_calc(y) # 3.
```

scope

Value of x at #1, #2 and #3?

```
x = 4
x *= 2              # 1.
def do_calc(x):
    print(x)        # 2.
    return x ** 2

do_calc(x)
y = x + 2
x = do_calc(y) # 3.
```

Ans:

1. 8
2. 8

scope

Value of x at #1, #2 and #3?

```
x = 4
x *= 2              # 1.
def do_calc(x):
    print(x)        # 2.
    return x ** 2

do_calc(x)
y = x + 2
x = do_calc(y) # 3.
```

Ans:

1. 8
2. 8 , 10
3. 100

function 1

```
def total_length(words):
    total = 0
    for word in words:
        total += len(word)
    return total

color = [”red”, ”green”, ”blue”]
lenX = total_length(color)
```

function 1

```
def total_length(words):
    total = 0
    for word in words:
        total += len(word)
    return total

color = [”red”, ”green”, ”blue”]
lenX = total_length(color)
```

lenX = 12

function 2

```
def word_lengths(words):
    lengths = ____
    k = 0
    for word in words:
        lengths ____
        k += 1
    return lengths

wlengths = word_lengths(["red","green","blue"])
```

How will you modify the code to get the answer as `wlengths = [3, 5, 4]`?

function 2

How to modify the code to get wlengths = [3, 5, 4]?

```
def word_lengths(words):
    lengths = [0]*len(words)
    k = 0
    for word in words:
        lengths[k] = len(word)
        k += 1
    return lengths
```

function 3

```
def fun( a ):
    return a + 2
    return a - 2

x = fun( 2 ) * fun( 3 )
```

What is the value of `x`?

A `6`

B `8`

C `24`

D None of the above.

function 3

```
def fun( a ):
    return a + 2
    return a - 2

x = fun( 2 ) * fun( 3 )
```

What is the value of `x`?

A `6`

B `8`

C `24`

D None of the above. ⋆ (`20`)

default in function

```
def funcName( xx = 99 )
    yy = xx + 1
    zz = xx + 2
    return yy, zz

att = funcName()
```

default in function

```
def funcName( xx = 99 )
    yy = xx + 1
    zz = xx + 2
    return yy, zz

att = funcName()
```

Ans: (100, 101)

```
att = funcName(1)
```

default in function

```
def funcName( xx = 99 )
    yy = xx + 1
    zz = xx + 2
    return yy, zz

att = funcName()
```

Ans: (100, 101)

```
att = funcName(1)
```

Ans: (2, 3)

```
att = funcName(xx=8)
```

default in function

```
def funcName( xx = 99 )
    yy = xx + 1
    zz = xx + 2
    return yy, zz

att = funcName()
```

Ans: (100, 101)

```
att = funcName(1)
```

Ans: (2, 3)

```
att = funcName(xx=8)
```

Ans: (9, 10)

```
att = funcName(yy=8)
```

default in function

```
def funcName( xx = 99 )
    yy = xx + 1
    zz = xx + 2
    return yy, zz

att = funcName()
```

Ans: (100, 101)

```
att = funcName(1)
```

Ans: (2, 3)

```
att = funcName(xx=8)
```

Ans: (9, 10)

```
att = funcName(yy=8)
```

Ans: Error

File Input & Output

Objectives

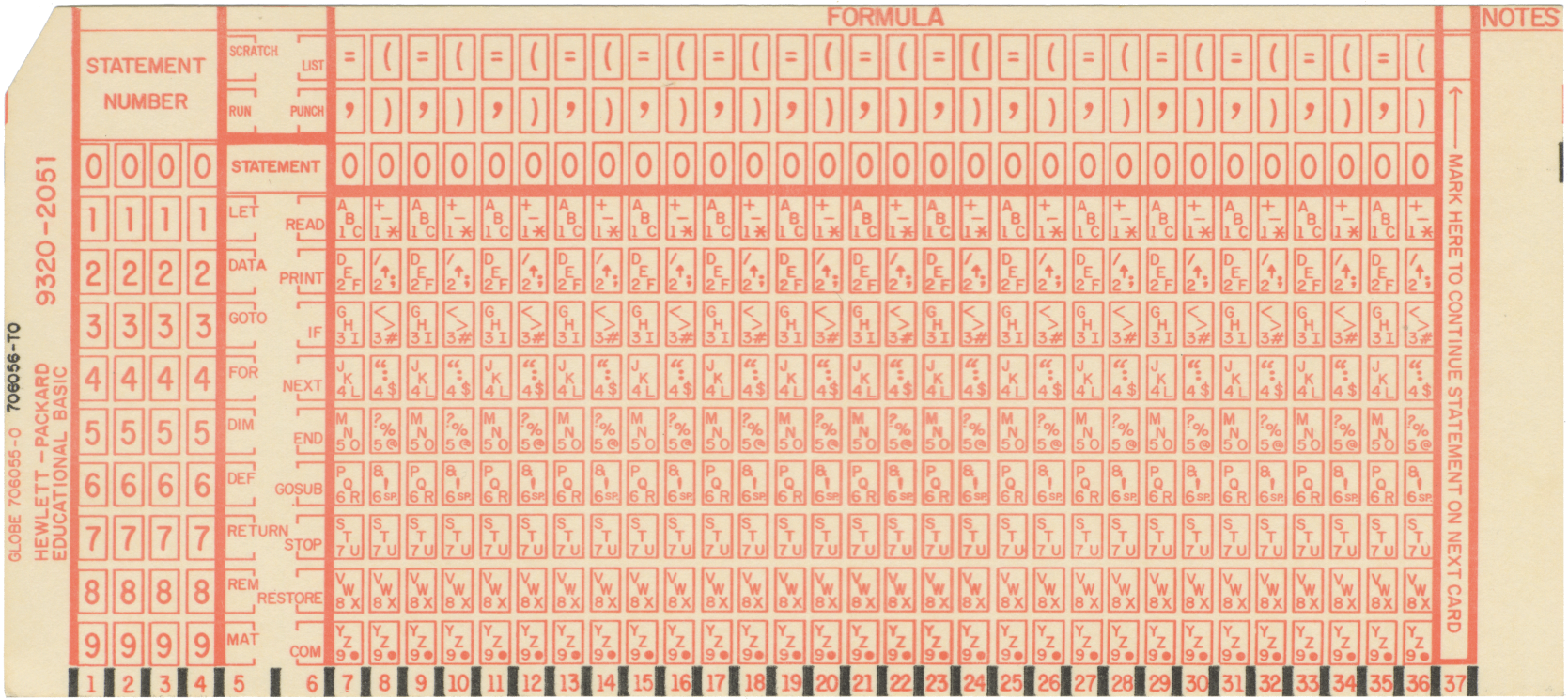
A. Access data stored in `files` as plain-text.
B. Use `loops` with multiple levels effectively.
C. Distinguish between the use of `read()` and `readlines()`.
D. Use multiple-level loops to read a file.
E. Use `split()` to logically divide data and `join()` to unite them.
F. Use loop aids like `break`, `continue`, `zip`, and `enumerate`.
G. Write a file.

Files

It is uncommon to generate the source data to be processed in the same program as one uses it.

What is a file?

Punch card


FORMULA
NOTES
STATEMENT NUMBER
SCRATCH
LIST
RUN
PUNCH
STATEMENT
LET
READ
DATA
PRINT
GOTO
IF
FOR
NEXT
DIM
END
DEF
GOSUB
RETURN
STOP
REM
RESTORE
MAT
COM
MARK HERE TO CONTINUE STATEMENT ON NEXT CARD
9320-2051
HEWLETT-PACKARD
EDUCATIONAL BASIC
GLOBE 706055-0 706056-TO

Punch card deck—5 MB

Secondary storage

M1. Files

`file` is an iterable data type created by the function `open`.

M1. Files

`file` is an iterable data type created by the function `open`.

=> `open` creates a data type called `file` which can be iterated.

M1. Files

`file` is an iterable data type created by the function `open`.

=> `open` creates a data type called `file` which can be iterated.

`open` accepts two options:

the first one is the *file name* as a *string*.
the second one is the *file mode* as a *string* that tells python what to do with the file. e.g., ”r”, ”w”, ”a”

Each item in the iterable is a `string` representing one line in the file.

```
myfile = open( 'wordlist.txt' , 'r')
for line in myfile:
    print( line )
```

- What data type is `line`?

Example 1

```
total = 0
for line in open( 'numbers.txt' ):
    total += int( line )
    #To have no error, what do you expect each "lin

print( total )
```

Example 2

```
for w in open( 'words.txt' , 'r'):
    vowels = 0

    for c in w.lower():
        if c in 'aeiou':
            vowels += 1

    print( w.strip() + ' %i' % vowels )
```

File workflow

If we `open` a `file`, we should `close` it as well.

`close` protects the file against data loss.

```
myfile = open( 'words.txt' , 'r')

for line in myfile:
    print( line )

myfile.close()  # process responsibly
```

M2. File read - .read()

The default way of opening a `file` is to `'r'`ead it.

We can *extract and store* all of the data from the file into *a variable* at once with:

`read`, which returns a string

```
myfile = open( 'words.txt' , 'r')
mydata = myfile.read()
myfile.close()

print( mydata )
```

M3. File read - .readlines()

The default way of opening a `file` is to `'r'`ead it.

We can extract all of the data from the file at once with:

`readlines`, which returns a `list` of strings. Each string contains one line in the file.

```
myfile = open( 'words.txt' , 'r')
mydata = myfile.readlines()
myfile.close()
for line in mydata:
    print( line )
```

File read

After reading, the `file` will be at the end of data.

Another `.read()` or `.readlines()` will return nothing (unless other commands are used, not in CS101)

Question

How different ways of reading a file?

Question

How different ways of reading a file?

```
file = open(”z.txt”,”r”)

1. file.read()
 => one string with everything in a file
2. file.readlines()
 => a list of strings where
            each string is a line in a file
```

Question

How different ways of reading a file?

```
file = open("z.txt","r")

1. file.read()
 => one string with everything in a file
2. file.readlines()
 => a list of strings where
              each string is a line in a file

3. for i in file:
 => access one line by one line in a file
```

What can you do after reading a file?

```
myfile = open( 'words.txt' , 'r')
mydata = myfile.readlines()
myfile.close()
for line in mydata:
    print( line )
```

Just `print()`? What data type is `mydata`?

String and List of strings

String data

The `string` is the same as the `string` data type that you already know

Common operators used: `.join()` and `.split()`

What kind of operators are these?

String.split() example

`.split()` : a string method that accepts a `str` to split by and returns a `list` of str.

```
my_string = ' ZJUI is more selfish than
     ZJU because of the "I" '
a = my_string.split( ' ' )
```

The sentence is split using the delimited ' '.
What are a and len(a)?

String.split() example

`.split()` : a string method that accepts a `str` to split by and returns a `list` of str.

```
my_string = ' ZJUI is more selfish than
     ZJU because of the "I" '
a = my_string.split( ' ' )
```

The sentence is split using the delimited ' '.
What are a and len(a)?

```
a = ['', 'ZJUI', 'is', 'more', 'selfish',
'than', 'ZJU', 'because', 'of', 'the',
'"I"', '']
```

String.split() example

`.split()` : a string method that accepts a `str` to split by and returns a `list` of str.

```
my_string = ' ZJUI is more selfish than
     ZJU because of the "I" '
a = my_string.split( ' ' )
```

The sentence is split using the delimited ' '.
What are a and len(a)?

```
a = ['', 'ZJUI', 'is', 'more', 'selfish',
'than', 'ZJU', 'because', 'of', 'the',
'"I"', '']

len(a) = 12
```

String.join() example

`.join()` : a `string method` that accepts a `list` of str and returns a `str`.

```
my_list = [ 'All','the','handsome','boys',
'and','pretty','girls','are','in','Year','1']

combine = ' '.join( my_list )
```

The sentence is formed by joining the individual words with ' '.

String.join() example

`.join()` : a `string method` that accepts a `list` of str and returns a `str`.

```
my_list = [ 'All','the','handsome','boys',
'and','pretty','girls','are','in','Year','1']

combine = ' '.join( my_list )
```

The sentence is formed by joining the individual words with ' '.

Ans:

```
combine = 'All the handsome boys and pretty
girls are in Year 1'
```

Multiple-level loops Question

If you have this file named ”menu.csv” containing:

Drinks, Size, Price

Latte, M, 10

Latte, L, 15

Tea, M, 8

Coke, M, 5

How will you print each item in this file?

Answer

```
d_file = open( 'kentucky-derby.csv','r' )
d_data = d_file.read()
d_file.close()
```

Answer

```
d_file = open( 'kentucky-derby.csv','r' )
d_data = d_file.read()
d_file.close()

rows = d_data.split( '\n' )
```

Answer

```
d_file = open( 'kentucky-derby.csv','r' )
d_data = d_file.read()
d_file.close()

rows = d_data.split( '\n' )

for row in rows:
    fields = row.split( ',' )
```

Answer

```
d_file = open( 'kentucky-derby.csv','r' )
d_data = d_file.read()
d_file.close()

rows = d_data.split( '\n' )

for row in rows:
    fields = row.split( ',' )
        for field in fields:
            print( field )
```

Loop Aids

Loop management: Loop Aids

`break` - stops the loop that `break` is immediately in

`continue` - skips and continues to the next iteration of the current loop

`zip` - iterates two lists at the same time

`enumerate` - gets the item and its position/index in a list

`permutations` - gives all possible sets of permutation

Loop management: break

```
i = 0
while i < 10:
    i += 1
    if i == 4:
        break  # terminate the loop
    print(i)
```

Ans:

Loop management: break

```
i = 0
while i < 10:
    i += 1
    if i == 4:
        break  # terminate the loop
    print(i)
```

Ans:
1
2
3

Loop management: continue

```
i = 0
while i < 10:
    i += 1
    if i == 4:
        continue  # skip ONLY this iteration
    print(i)
```

Ans:

Loop management: continue

```
i = 0
while i < 10:
    i += 1
    if i == 4:
        continue  # skip ONLY this iteration
    print(i)
```

Ans:
1
2
3
5
...
10

Accessing lists - zip

Sometimes we have two `list`s that correspond to each other.

If we want to loop over both together, we have two approaches open:

```
qs = [ 'name', 'quest', 'favourite colour' ]
as = [ 'Meimei', 'Have fun', 'Fun color' ]

# M1:
for i in range(len(qs)):
    print( 'What is your %s?
                It is %s.'%(qs[i],as[i]) )
```

Accessing lists - zip

Sometimes we have two `list`s that correspond to each other.

If we want to loop over both together, we have two approaches open:

```
qs = [ 'name', 'quest', 'favourite colour' ]
as = [ 'Meimei', 'Have fun', 'Fun color' ]

# M1:
for i in range(len(qs)):
    print( 'What is your %s?
                It is %s.'%(qs[i],as[i]) )

# M2:
for q,a in zip(qs,as):
    print( 'What is your %s?  It is %s.'%(q,a) )
```

Accessing lists - zip

`zip` makes two `list`s *jointly iterable*.

```
def pick( a,b ):
    result = [ ]  # a list of values

    for i,j in zip(a,b):
        result.append( i+j )

    return result
```

Accessing lists - zip

`zip` makes two `list`s *jointly iterable*.

```
def pick( a,b ):
    result = [ ]  # a list of values

    for i,j in zip(a,b):
        result.append( i+j )

    return result
```

What if `len(a)=6` and `len(b)=10`? How many loops will `zip` perform?

Accessing lists - zip

`zip` makes two `list`s *jointly iterable*.

```
def pick( a,b ):
    result = [ ]  # a list of values

    for i,j in zip(a,b):
        result.append( i+j )

    return result
```

What if `len(a)=6` and `len(b)=10`? How many loops will `zip` perform?
Ans: 6 loops

Accessing lists - enumerate

What if you need to know both the *value* and the *index* of the item?

```
my_list = [ 'meter', 'kilogram', 'second' ]

# M1
for i in range( len(my_list) ):
    print( '%s is the %sth item.' % (my_list[i],i)
```

Accessing lists - enumerate

What if you need to know both the *value* and the *index* of the item?

```
my_list = [ 'meter', 'kilogram', 'second' ]

# M1
for i in range( len(my_list) ):
    print( '%s is the %sth item.' % (my_list[i],i)

# M2
for i,item in enumerate( my_list ):
    print( '%s is the %sth item.' % (item,i) )
```

Accessing lists

Both `zip` and `enumerate` are *convenience* functions!

There are multiple approaches!

Other File-related stuff

File modes

We can also 'w'rite to a file, but we need to open it differently.

We can specify a file mode when we open a file:

- 'r' to read a file's data (default)
- 'w' to write data to a file

```
myfile = open( 'words.txt','w' )
myfile.write( 'Hello, this is a test.' )
myfile.close()  # ultra-important now!
```

Other modes available but not important for 101.

Question

Assume a.txt contains:
abc
123
Sesame

```
file1 = open(”a.txt”,”w”)
data1 = file1.readines()
file1.close()
print(data1)
```

What will be printed?

Question

Assume a.txt contains:
abc
123
Sesame

```
file1 = open(”a.txt”,”w”)
data1 = file1.readines()
file1.close()
print(data1)
```

What will be printed?
Ans:
Error!

File paths

Windows refers to files from `C:\`.

Linux (and Unix, including macOS) refers to files from `/`.

Sometimes in Windows, `/` can be used in place of `C:\`.

File paths

The *file path* describes where a file can be found on the file system.

- Relative paths start from where you are (same directory).
- Absolute paths start from the system root (start with `/` or `C:\\`
- There are two special ”paths”: `.` (the current directory) and `..` (the parent directory).

File paths

You have a file called 'lab01.ipynb' at

```
c:\home\netid\cs101-sp18\
```

To open this file, use:
absolute path - regardless where you currently are

```
1. 'c:\\home\\netid\\cs101-sp18\\lab01.ipynb'
2. '/home/netid/cs101-sp18/lab01.ipynb'
```

relative path If you are already @ /home/netid/

```
1. './cs101-sp18/lab01.ipynb'
2. 'cs101-sp18/lab01.ipynb'
3. '../netid/cs101-sp18/lab01.ipynb'
```

Summary

Summary

1. `file` type is iterable => can be used in `for` loop
2. `open(xxx,yyy)` file with `'r'` or `'w'`
3. 3 different ways to read a file
4. `split('something'), 'something'.join()`
5. Loop Aids: `break, continue, zip, enumerate`
6. File Path
7. `.close()`