

Python

Control Flow

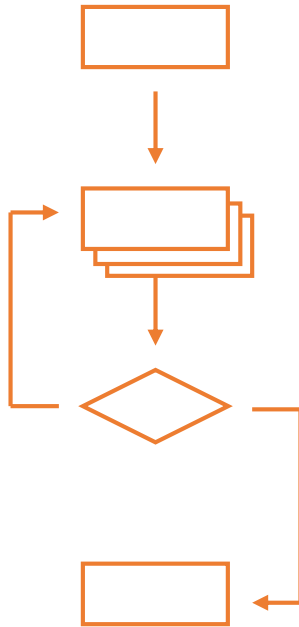
Real power of programs comes from:

Real power of programs comes from:

repetition

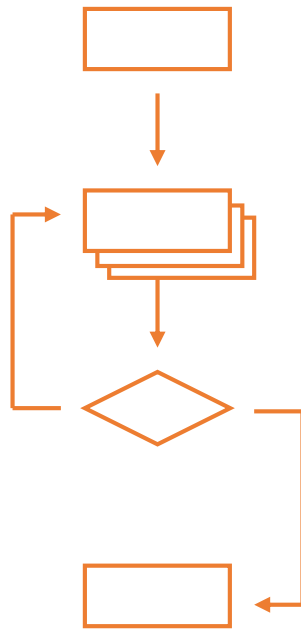
Real power of programs comes from:

repetition



Real power of programs comes from:

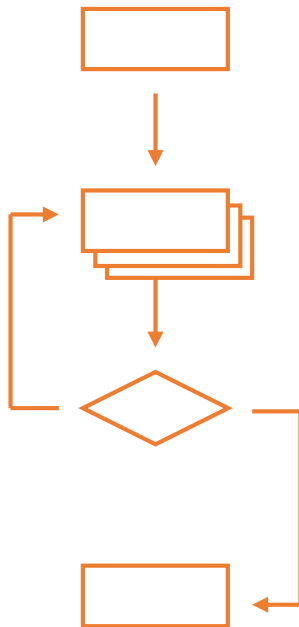
repetition



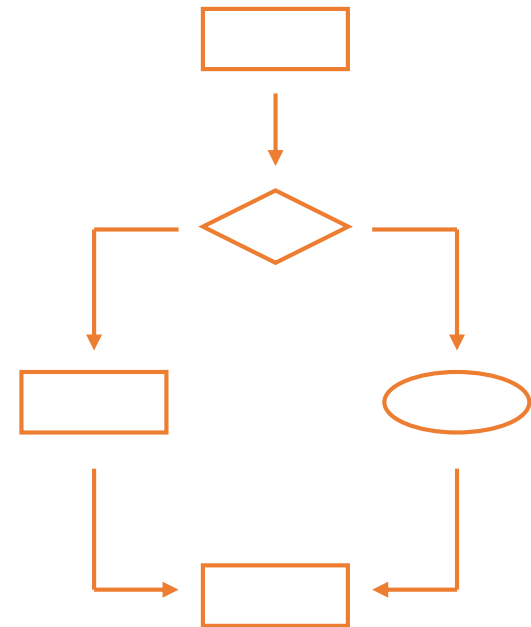
selection

Real power of programs comes from:

repetition



selection



Simplest form of repetition is *while loop*

Simplest form of repetition is *while loop*

```
num_moons = 3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
```


Simplest form of repetition is *while loop*

```
num_moons = 3
while num_moons > 0: ← test
    print(num_moons)
    num_moons -= 1
```

Simplest form of repetition is *while loop*

```
num_moons = 3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
```

← do

Simplest form of repetition is *while loop*

```
num_moons = 3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
```

3

← do

Simplest form of repetition is *while loop*

```
num_moons = 3
while num_moons > 0: ← test again
    print(num_moons)
    num_moons -= 1
```

3

Simplest form of repetition is *while loop*

```
num_moons = 3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
```

3

2

Simplest form of repetition is *while loop*

```
num_moons = 3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
```

3

2

1

While loop may execute zero times

While loop may execute zero times

```
print('before')
num_moons = -3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
print('after')
```


While loop may execute zero times

```
print('before')  
num_moons = -3  
while num_moons > 0: ← not true when first tested...  
    print(num_moons)  
    num_moons -= 1  
print('after')
```

While loop may execute zero times

```
print('before')
num_moons = -3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
print('after')
```

← ...so this is never executed

While loop may execute zero times

```
print('before')  
num_moons = -3  
while num_moons > 0:  
    print(num_moons)  
    num_moons -= 1  
print('after')  
before  
after
```

While loop may execute zero times

```
print('before')
num_moons = -3
while num_moons > 0:
    print(num_moons)
    num_moons -= 1
print('after')
```

before
after

Important to consider this case when designing
and testing code

While loop may also execute forever

While loop may also execute forever

```
print('before')  
num_moons = 3  
while num_moons > 0:  
    print(num_moons)  
print('after')
```

While loop may also execute forever

```
print('before')
num_moons = 3
while num_moons > 0:
    print(num_moons)
print('after')
before
```

While loop may also execute forever

```
print('before')  
num_moons = 3  
while num_moons > 0:  
    print(num_moons)  
print('after')  
before  
3
```


While loop may also execute forever

```
print('before')
num_moons = 3
while num_moons > 0:
    print(num_moons)
print('after')
```

before

3

3

While loop may also execute forever

```
print('before')
num_moons = 3
while num_moons > 0:
    print(num_moons)
print('after')
```

before

3

3

3

While loop may also execute forever

```
print('before')
num_moons = 3
while num_moons > 0:
    print(num_moons)
print('after')
```

before

3

3

3

⋮

While loop may also execute forever

```
print('before')
```

```
num_moons = 3
```

```
while num_moons > 0:
```

```
    print(num_moons)
```

```
print('after')
```

```
before
```

```
3
```

```
3
```

```
3
```

```
⋮
```

← Nothing in here changes
the loop control condition

While loop may also execute forever

```
print('before')
num_moons = 3
while num_moons > 0:
    print(num_moons)
print('after')
```

before

3

3

3

⋮

Usually not the desired behavior...

While loop may also execute forever

```
print('before')
num_moons = 3
while num_moons > 0:
    print(num_moons)
print('after')
```

before
3
3
3
⋮

Usually not the desired behavior...

...but there *are* cases where it's useful

Why indentation?

Why indentation?

Studies show that's what people actually pay
attention to

Why indentation?

Studies show that's what people actually pay attention to

- Every textbook on C or Java has examples where indentation and braces don't match

Why indentation?

Studies show that's what people actually pay attention to

- Every textbook on C or Java has examples where indentation and braces don't match

Doesn't matter how much you use, but whole block must be consistent

Why indentation?

Studies show that's what people actually pay attention to

- Every textbook on C or Java has examples where indentation and braces don't match

Doesn't matter how much you use, but whole block must be consistent

Python Style Guide (PEP 8) recommends 4 spaces

Why indentation?

Studies show that's what people actually pay attention to

- Every textbook on C or Java has examples where indentation and braces don't match

Doesn't matter how much you use, but whole block must be consistent

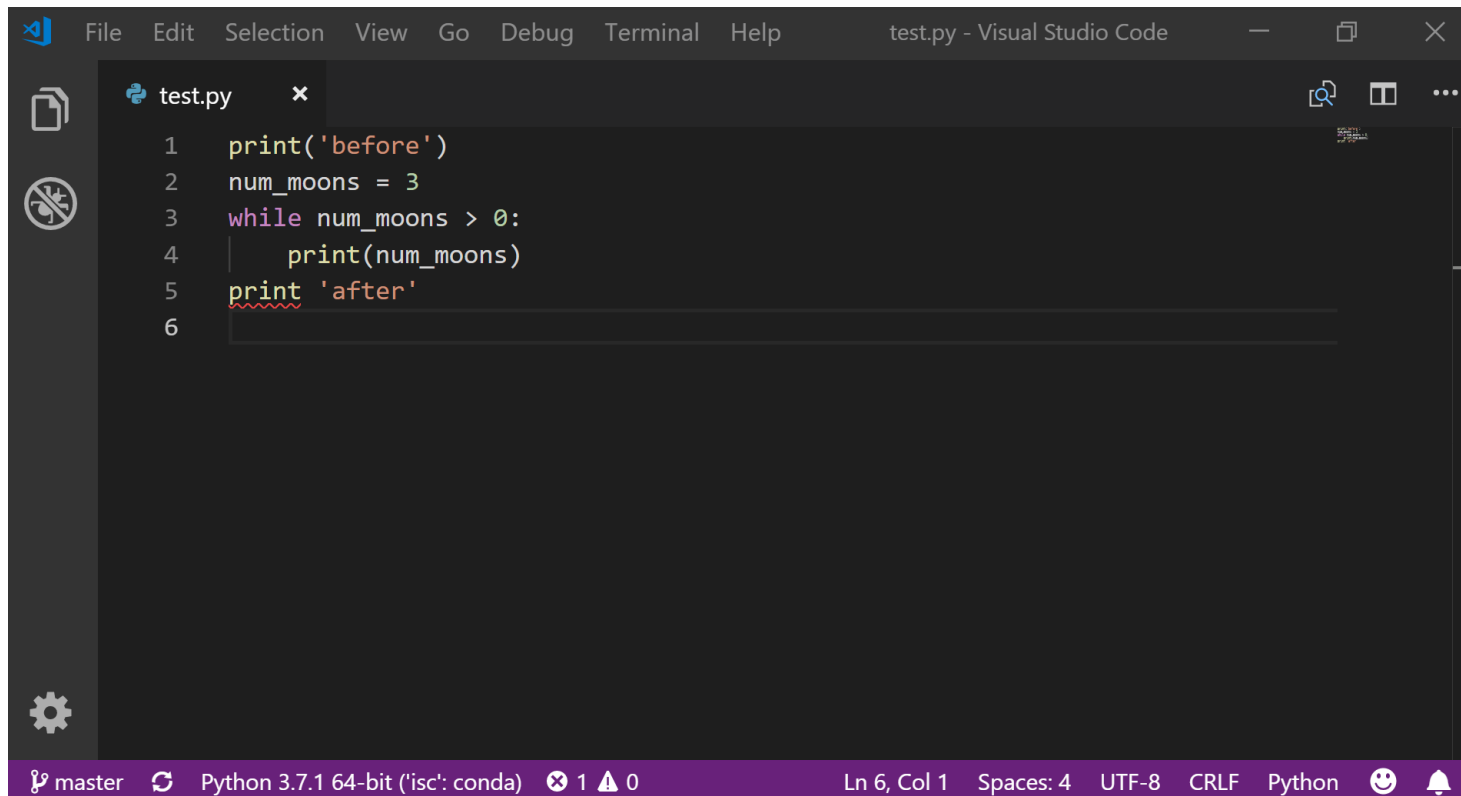
Python Style Guide (PEP 8) recommends 4 spaces

And no tab characters

Side note on IDEs (Integrated Development Environments)

Side note on IDEs (Integrated Development Environments)

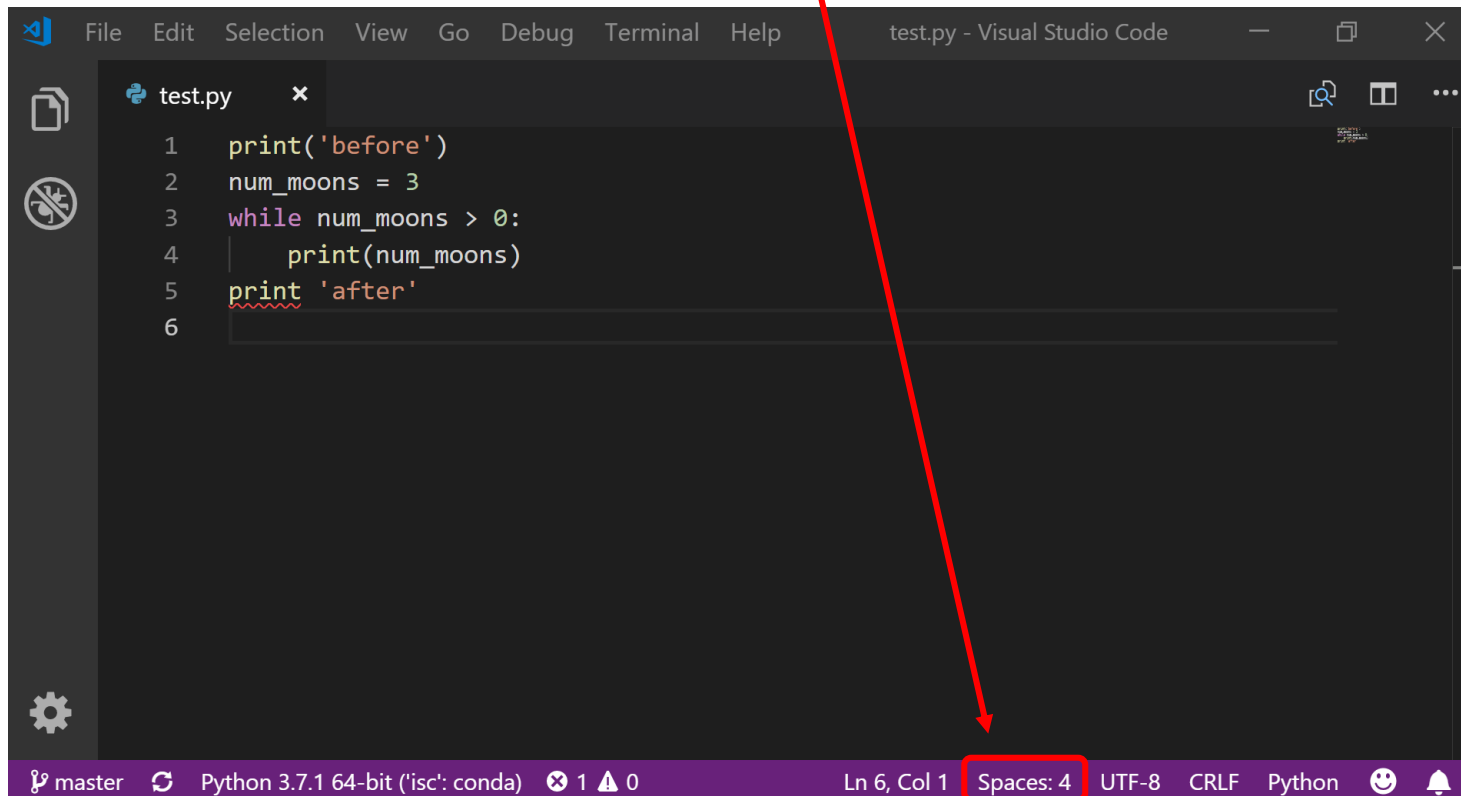
An IDE is a nicer place to write, edit and run code from all in one. Most often also include syntax highlighting, linting and debugging built in (debugging will be taught later in the course).



```
test.py
1  print('before')
2  num_moons = 3
3  while num_moons > 0:
4      print(num_moons)
5  print 'after'
6
```

Side note on IDEs (Integrated Development Environments)

Most IDEs will also let you choose your indentation too, so you don't have to manually type 4 spaces...



```
test.py
1 print('before')
2 num_moons = 3
3 while num_moons > 0:
4     print(num_moons)
5 print 'after'
6
```

master Python 3.7.1 64-bit ('isc': conda) 1 0 Ln 6, Col 1 **Spaces: 4** UTF-8 CRLF Python

Use `if`, `elif`, and `else` to make choices

Use if, elif, and else to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
```

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:                ← not true when first tested...
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
```

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less') ← ...so this is not executed
elif moons == 0:
    print('equal')
else:
    print('greater')
```

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0: ← this isn't true either...
    print('equal')
else:
    print('greater')
```

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal') ← ...so this isn't executed
else:
    print('greater')
```

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
```

← nothing else has executed...

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater') ← ...so this is executed
```

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
greater
```


Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
greater
```

Always start with **`if`**

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
greater
```

Always start with **`if`**

Can have any number of **`elif`** clauses (including none)

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
greater
```

Always start with **`if`**

Can have any number of **`elif`** clauses (including none)

And the **`else`** clause is optional

Use `if`, `elif`, and `else` to make choices

```
moons = 3
if moons < 0:
    print('less')
elif moons == 0:
    print('equal')
else:
    print('greater')
greater
```

Always start with **`if`**

Can have any number of **`elif`** clauses (including none)

And the **`else`** clause is optional

Always tested in order

Blocks may contain blocks

Blocks may contain blocks

```
num = 0
while num <= 10:
    if (num % 2) == 1:
        print(num)
    num += 1
```

Blocks may contain blocks

```
num = 0
while num <= 10:
    if (num % 2) == 1:
        print(num)
    num += 1
```

Count from 0 to 10



Blocks may contain blocks

```
num = 0
while num <= 10:
    if (num % 2) == 1:
        print(num) ← Print odd numbers
    num += 1
```


Blocks may contain blocks

```
num = 0
while num <= 10:
    if (num % 2) == 1:
        print(num)
    num += 1
```

1
3
5
7
9

A better way to do it

A better way to do it

```
num = 1
while num <= 10:
    print(num)
    num += 2
```

A better way to do it

```
num = 1
while num <= 10:
    print(num)
    num += 2
```

1
3
5
7
9

Stop here

Print primes less than 1000

Print primes less than 1000

```
num = 2
while num <= 1000:
    ...figure out if num is prime...
    if is_prime:
        print(num)
    num += 1
```

Print primes less than 1000

```
num = 2
while num <= 1000:
    ...figure out if num is prime...
    if is_prime:
        print(num)
    num += 1
```

Cannot be evenly divided
by any other integer

Print primes less than 1000

```
num = 2
while num <= 1000:
    ...figure out if num is prime...
    if is_prime:
        print(num)
    num += 1
```

↙

```
is_prime = True
trial = 2
while trial < num:
    if ...num divisible by trial...:
        is_prime = False
    trial += 1
```

Print primes less than 1000

```
num = 2
while num <= 1000:
    ...figure out if num is prime...
    if is_prime:
        print(num)
    num += 1
```

is_prime = **True**

trial = 2

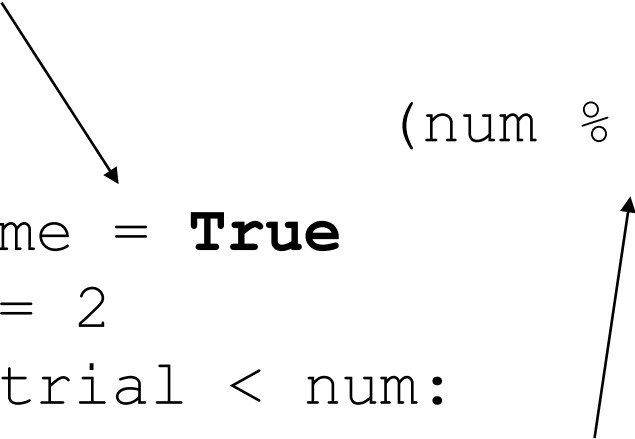
```
while trial < num:
    if ...num divisible by trial...:
        is_prime = False
    trial += 1
```

Remainder is zero

Print primes less than 1000

```
num = 2
while num <= 1000:
    ...figure out if num is prime...
    if is_prime:
        print(num)
    num += 1
    is_prime = True
    trial = 2
    while trial < num:
        if ...num divisible by trial...:
            is_prime = False
        trial += 1
```

(num % trial) == 0



Print primes less than 1000

```
num = 2
while num <= 1000:
    is_prime = True
    trial = 2
    while trial < num:
        if (num % trial) == 0:
            is_prime = False
            trial += 1
    if is_prime:
        print(num)
    num += 1
```

A more efficient way to do it

A more efficient way to do it

```
num = 2
while num <= 1000:
    is_prime = True
    trial = 2
    while trial**2 < num:
        if (num % trial) == 0:
            is_prime = False
        trial += 1
    if is_prime:
        print(num)
    num += 1
```

A more efficient way to do it

```
num = 2
while num <= 1000:
    is_prime = True
    trial = 2
    while trial**2 < num:
        if (num % trial) == 0:
            is_prime = False
            trial += 1
        else:
            trial += 1
    if is_prime:
        print(num)
    num += 1
```

← N cannot be divided
evenly by any number
greater than \sqrt{N}

Any code that hasn't been tested is probably wrong

Any code that hasn't been tested is probably wrong

```
num = 2
while num <= 10:
    is_prime = True
    trial = 2
    while trial**2 < num:
        if (num % trial) == 0:
            is_prime = False
        trial += 1
    if is_prime:
        print(num)
    num += 1
```

Any code that hasn't been tested is probably wrong

<pre>num = 2</pre>	2
<pre>while num <= 10:</pre>	3
<pre> is_prime = True</pre>	4
<pre> trial = 2</pre>	5
<pre> while trial**2 < num:</pre>	7
<pre> if (num % trial) == 0:</pre>	9
<pre> is_prime = False</pre>	
<pre> trial += 1</pre>	
<pre> if is_prime:</pre>	
<pre> print(num)</pre>	
<pre> num += 1</pre>	

Any code that hasn't been tested is probably wrong

```
num = 2
while num <= 10:
    is_prime = True
    trial = 2
    while trial**2 < num:
        if (num % trial) == 0:
            is_prime = False
        trial += 1
    if is_prime:
        print(num)
    num += 1
```

2

3

4

5

7

9

Any code that hasn't been tested is probably wrong

<pre>num = 2</pre>	2
<pre>while num <= 10:</pre>	3
<pre> is_prime = True</pre>	4
<pre> trial = 2</pre>	5
<pre> while trial**2 < num:</pre>	7
<pre> if (num % trial) == 0:</pre>	9
<pre> is_prime = False</pre>	
<pre> trial += 1</pre>	
<pre> if is_prime:</pre>	
<pre> print(num)</pre>	
<pre> num += 1</pre>	

Where's the bug?

Failures occur for perfect squares

Failures occur for perfect squares

```
num = 2
while num <= 10:
    is_prime = True
    trial = 2
    while trial**2 < num:
        if (num % trial) == 0:
            is_prime = False
        trial += 1
    if is_prime:
        print(num)
    num += 1
```

Failures occur for perfect squares

```
num = 2
while num <= 10:
    is_prime = True
    trial = 2
    while trial**2 < num: ← 2**2 == 4
        if (num % trial) == 0:
            is_prime = False
        trial += 1
    if is_prime:
        print(num)
    num += 1
```

Failures occur for perfect squares

```
num = 2
while num <= 10:
    is_prime = True
    trial = 2
    while trial**2 < num:
        if (num % trial) == 0:
            is_prime = False
            trial += 1
    if is_prime:
        print(num)
    num += 1
```

$2**2 == 4$

So never check to see

if $4 \% 2 == 0$

Failures occur for perfect squares

```
num = 2
while num <= 10:
    is_prime = True
    trial = 2
    while trial**2 < num:
        if (num % trial) == 0:
            is_prime = False
            trial += 1
    if is_prime:
        print(num)
    num += 1
```

$2**2 == 4$

So never check to see

if $4 \% 2 == 0$

Or if $9 \% 3 == 0$, etc.

More ways to control flow while inside a loop:

`break, continue, pass`

More ways to control flow while inside a loop:

`break, continue, pass`

e.g. Print the first multiple of a given value

break, continue, pass

e.g. Print the first multiple of a given value

```
value = 14
trial = 2
while trial < value:
    if trial % value == 0:
        print(trial)
        break
    trial += 1
```

2

break, continue, pass

e.g. Print the first odd multiple of a given value

```
value = 14
trial = 2
while trial < value:
    if trial % 2 == 0:
        trial += 1
        continue
    if trial % value == 0:
        print(trial)
        break
    trial += 1
```

7

break, continue, pass

If we aren't sure what we want to type yet but don't want the code to fail when ran we can type `pass` which does nothing

```
value = 14
trial = 2
while trial < value:
    if trial % 2 == 0:
        pass
    if trial % value == 0:
        print(trial)
        break
    trial += 1
```

2



created by

Greg Wilson

September 2010



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.