Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Python

Control Flow

Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

software carpentry

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

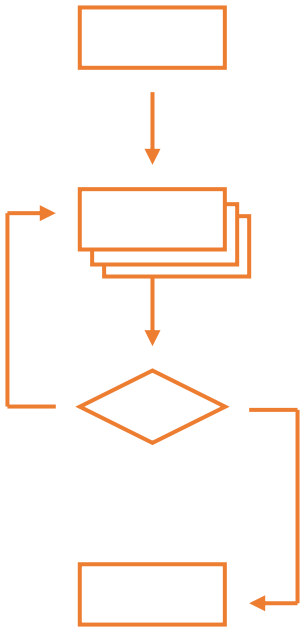
Real power of programs comes from:

Real power of programs comes from:

repetition

Real power of programs comes from:

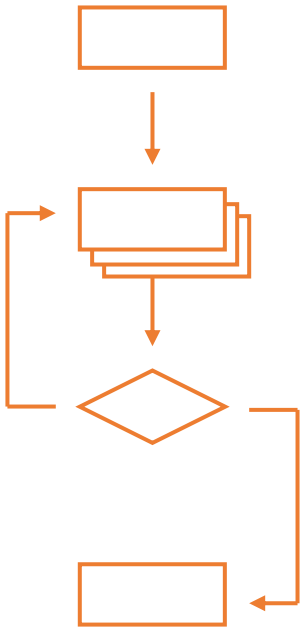
repetition



Real power of programs comes from:

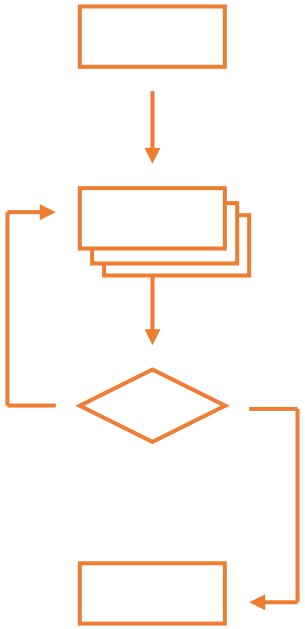
repetition

selection

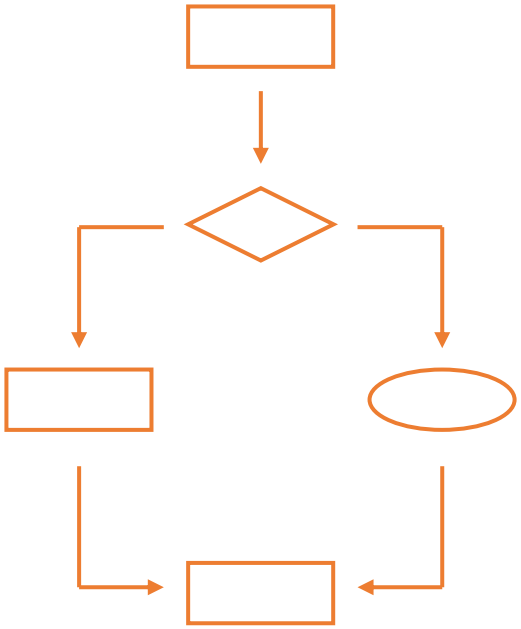


Real power of programs comes from:

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:
    print(num_moons)
    num_moons -= 1
```

test

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:
    print(num_moons)
    num_moons -= 1
```

3

do

Simplest form of repetition is *while loop*

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

← test again

```
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:
    print(num_moons)
    num_moons -= 1
print('after')
```

← not true when first tested…

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

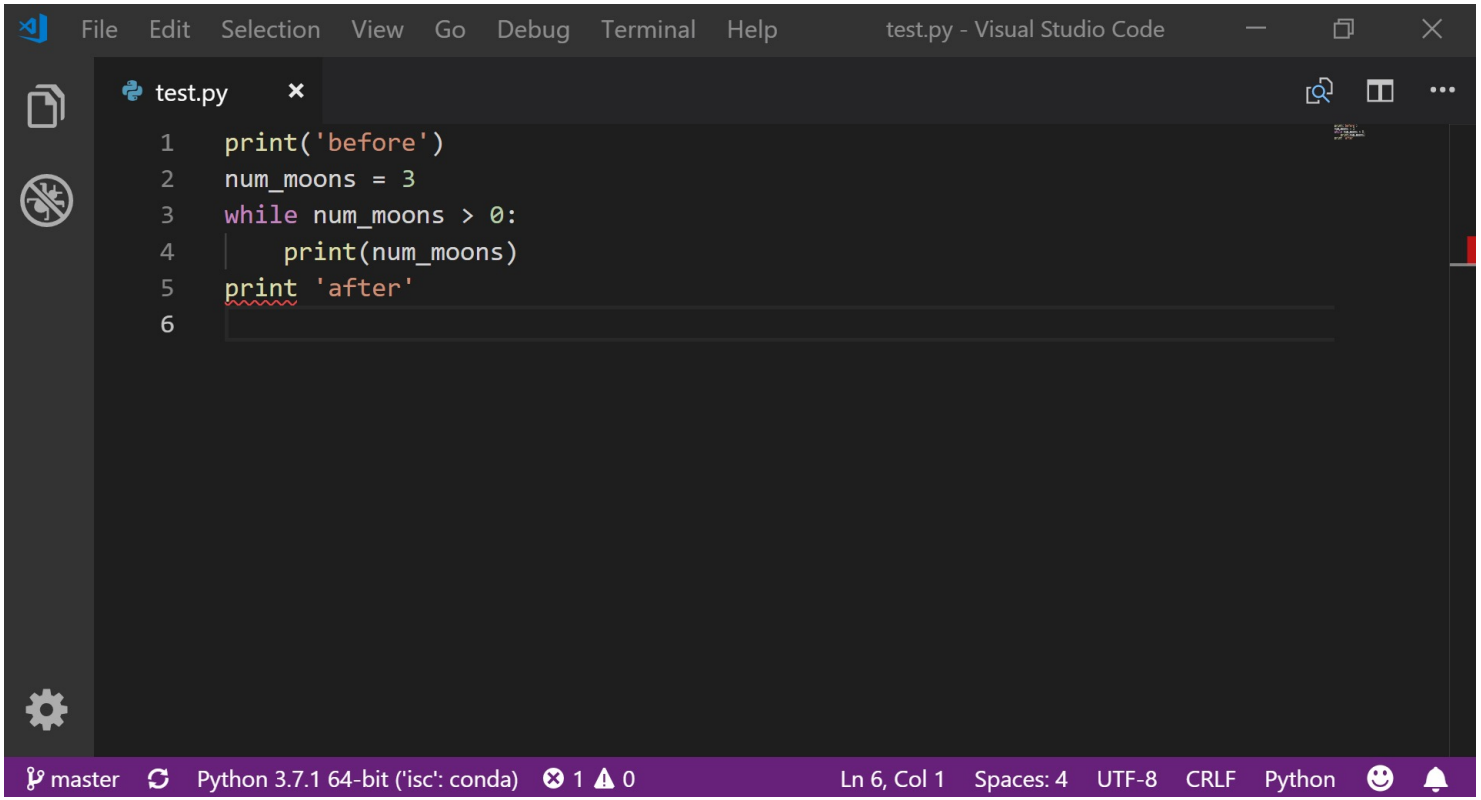
Python Style Guide (PEP 8) recommends 4 spaces

And no tab characters

Side note on IDEs (Integrated Development Environments)

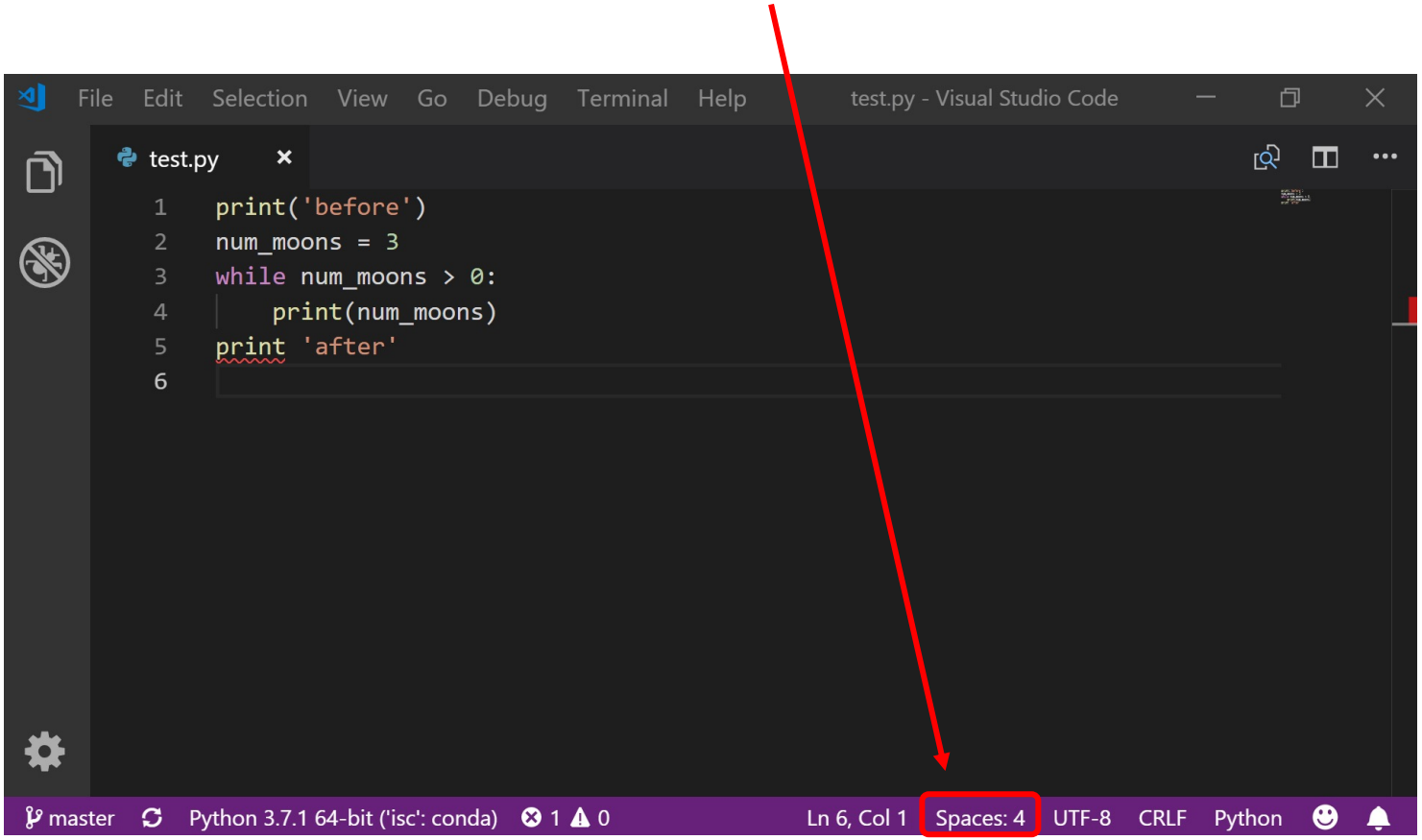
Side note on IDEs (Integrated Development Environments)

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



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…



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

← not true when first tested…

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 *not* executed

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

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

← this isn't true either...

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 isn't executed

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

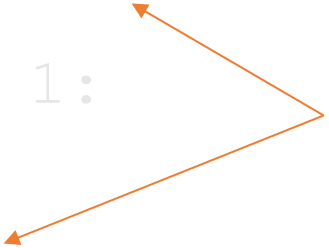
Blocks may contain blocks

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)
    num += 1
```

Print odd numbers

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
```

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 value % trial == 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 value % trial == 0:
        print(trial)
        break
    trial += 1
```

```
7
```

`break`, `continue`, `pass`

If you get to a point in your logic where you want to specifically do nothing, you can use `pass`

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

```
2
```

created by

Greg Wilson

September 2010