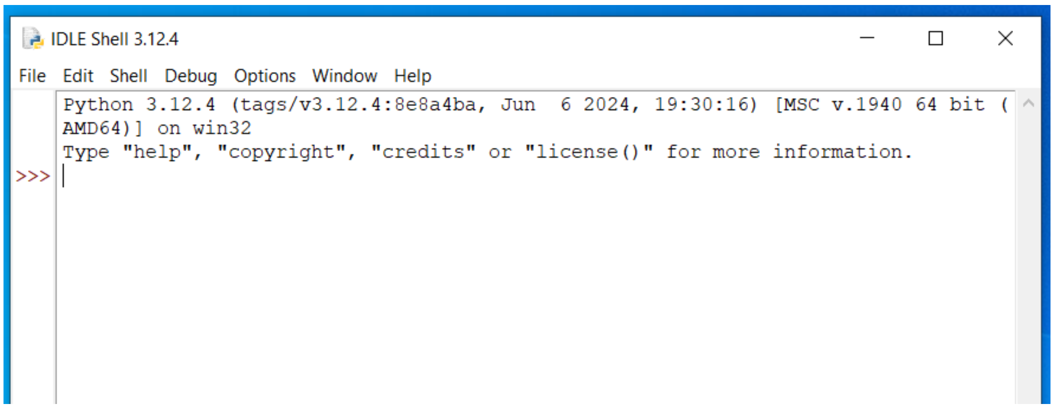
Introduction to Programming/ Week 1: Python interactive shell/ Maths in the shell

Maths in the shell

Load IDLE Shell

1. Open IDLE. The easiest way is to click the Start menu and search for 'IDLE'.
2. This should show you a IDLE Shell.



```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

Do calculations

The IDLE Shell will take commands and act like an advanced calculator.

For example, you can do basic calculations. Type a formula involving numbers and it will reply with the answer. You can use `+` and `-` as well as `*` for multiplication and `\` for division. For example, you could try this and Python should respond with the answer.

```
2*(5+7)
```

There are two main ways computers understand powers. Some programs use `^`, as in `2^6`, and others use `**`, as in `2**6`. Python uses `**`. Try this

```
(5+4)**2*4/2
```

One command at a time

Sometimes on this page, I might give several commands. You must run these one at a time. (We'll deal with writing programs, where a sequences of commands are given all together, later.)

For example, if you copy all these lines into the IDLE Shell together, you will get an error `SyntaxError: multiple statements found while compiling a single statement`. Instead, copy across one at a time.

```
a = 12
b = 18
a*b
```

Get setup to do advanced maths

1. Type or copy this command and press enter to load maths options. Nothing should happen, Python just runs your command and produces no output. It may take a little while to come back to the stage where you can enter the next command, which is when you see the >>> prompt.

   ```
   from sympy import *
   ```

2. Type or copy this command and press enter to define common letters that we might want to use as variable names. Again there is no output.

   ```
   from sympy.abc import x,y,z,t,n
   ```

Evaluating an expression

We can define an expression using the variable names we have told Python about. Type and run this command to define a function function $f(x) = x^7 + 3x^5 + 9$. There is no response at this stage.

```
f = x**7 + 3 * x**5 + 9
```

Now we can evaluate $f(x)$ at particular values of x. We do this using a command `subs()` to substitute values into our function. For example, type and run this command to evaluate $f(15)$.

```
f.subs(x, 15)
```

Sometimes, Python keeps things in exact form rather than giving a decimal number. For example, here we ask it to substitute $x = 2$ into the expression $\sqrt{x}$ and it just says `sqrt(2)`. Runs these commands one at a time.

```
s = sqrt(x)
s.subs(x,2)
```

We can get a decimal number from a SymPy expression using `.evalf()`. For example, here is a version of the example above where we ask for a decimal form at the end.

```
s.subs(x,2).evalf()
```

Simplifying expressions

One use of a symbolic computation package is to simplify mathematical expressions, which we can do in SymPy using `simplify()`. Here we simplify

$$\cos(x)\sin^2(x) + \frac{x^3+x^2-x}{x^2+x-1} + \cos^3(x)$$

```
simplify(cos(x)*sin(x)**2 + (x**3 + x**2 - x)/(x**2 + x - 1) + cos(x)**3)
```

One useful thing simplification can do for us is tell us whether two expressions are algebraically equivalent. Take, for example, $(2x+3)^2$ and $4x^2+12x+9$. If we simplify $(2x+3)^2-(4x^2+12x+9)$, then if these expressions are the same we should get 0.

```
simplify((2*x + 3)**2 - (4*x**2 + 12*x + 9))
```

Expanding expressions

We can expand expressions using `expand()`. For example, here we expand $(x^9-2)^4(x^4+x^2+7)$.

```
expand((x**9-2)**4*(x**4+x**2+7))
```

Solving equations

SymPy has a command called `solve()` which takes an expression and puts it equal to zero, then tries to solve it.

Here is a simple example - this takes $\sin(x)$ as the input and puts this equal to zero, so the equation it is trying to solve is $\sin(x)=0$.

```
solve(sin(x))
```

If you want to solve an equation that is not equal to zero, the simplest thing to do might be to rearrange it for zero. For example, if you want to solve $x^2=256$, ask `solve` to solve $x^2-256=0$.

```
solve(x**2-256)
```

You can use `solve` to combine powers in an expression. For example, to find a value for n in 2^n that equals $2^3 \times 2^5 \times 2^7$, run this code.

```
solve(2**3*2**5*2**7-2**n)
```

We can solve systems of equations by passing them in a list to the `solve` function. Take the following system of equations:

$$\begin{aligned} 3x + 2y - 4z &= -27; \\ 8x + 15y - 3z &= 1; \\ 5x + 4y + z &= 10. \end{aligned}$$

We can solve this system using `solve` - remember we rearrange each equation to be equal to zero. The list is in square brackets `[...]` with commas `,` between items.

```
solve([3*x+2*y-4*z+27,8*x+15*y-3*z-1,5*x+4*y+z-10])
```