

# Using IDLE





# Basic use of IDLE

---

- Start IDLE
  - You can type simple expressions and code directly into the “shell” window that comes up
  - You can use the up-arrow to go back and repeat previous lines
  - This is convenient for trying things out, or to use as a calculator, but not good for writing entire programs
- Create a new program (with the **New File** menu item, *control-N*) or open an existing program (with the **Open...** menu item, *control-O*)
  - There is also a **Recent Files** menu item you can use
- Edit the program, then run it using **Run > Run Module**, or just hit the **F5** key
  - Notice that some of the menus are different from those in the shell window



# Setting edit preferences

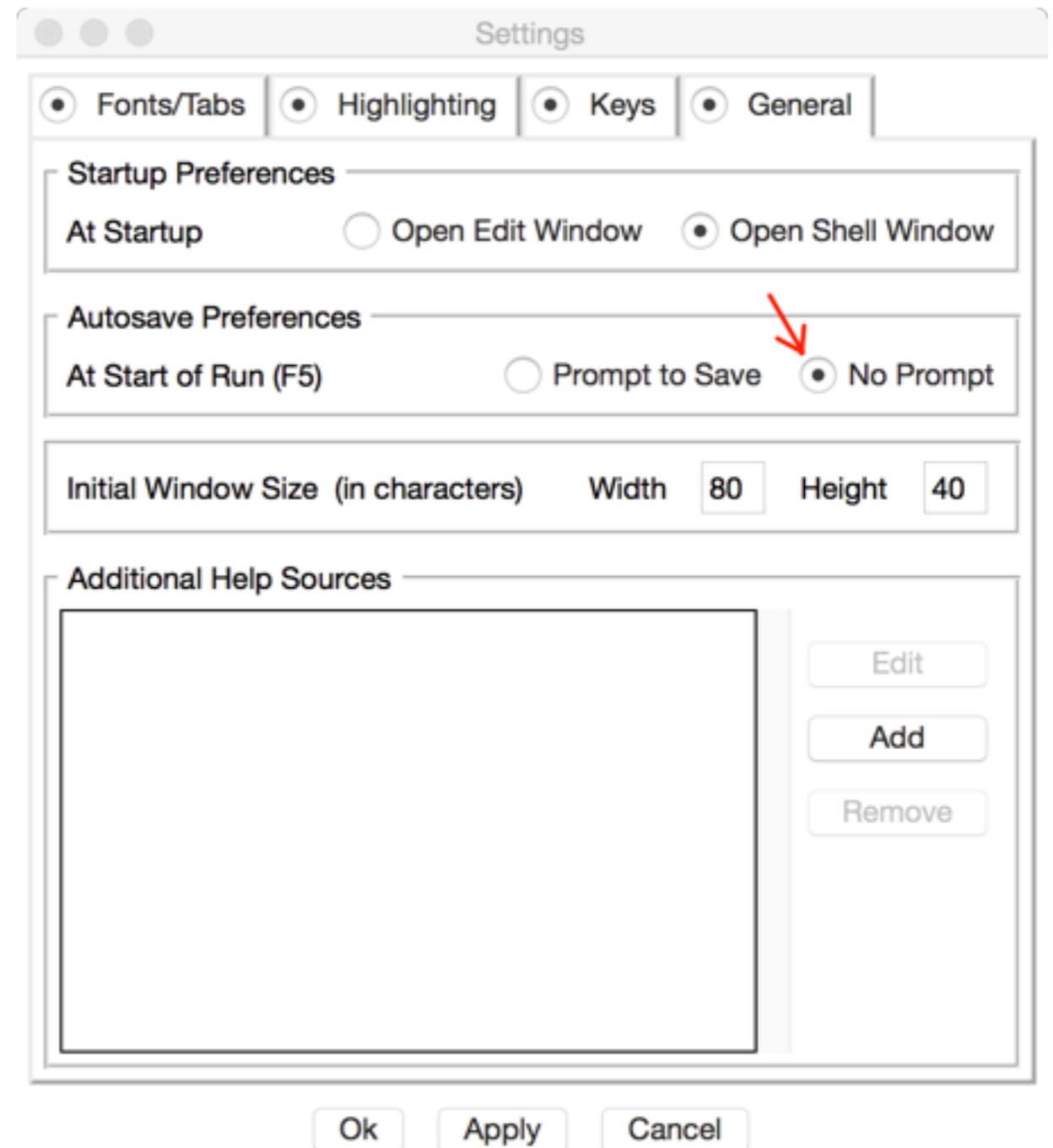
---

- The font you use can help or hinder your programming
  - It's important to distinguish **0** from **o**, **1** from **I** and **l** and **|** and **!**, **()** from **[]** and **{}**
    - Here are these same characters in Courier New:  
0 o 1 I l | ! ( ) [ ] { }
  - Also, it's helpful if punctuation marks are emphasized, especially **. , : ;**
  - My favorite font is Consolas; see <http://www.slant.co/topics/67/~what-are-the-best-programming-fonts> for some suggestions
- In programming, *a tab is an actual character*, representing some undefined but settable number of spaces
  - By default, when you hit the tab key, IDLE puts in four spaces, not an actual tab
  - If you use *any other editor* for your programs, be sure to do the same!



# Autosave

- When you edit a program and hit F5 to run it, IDLE asks you if you want to save first
- I find this annoying



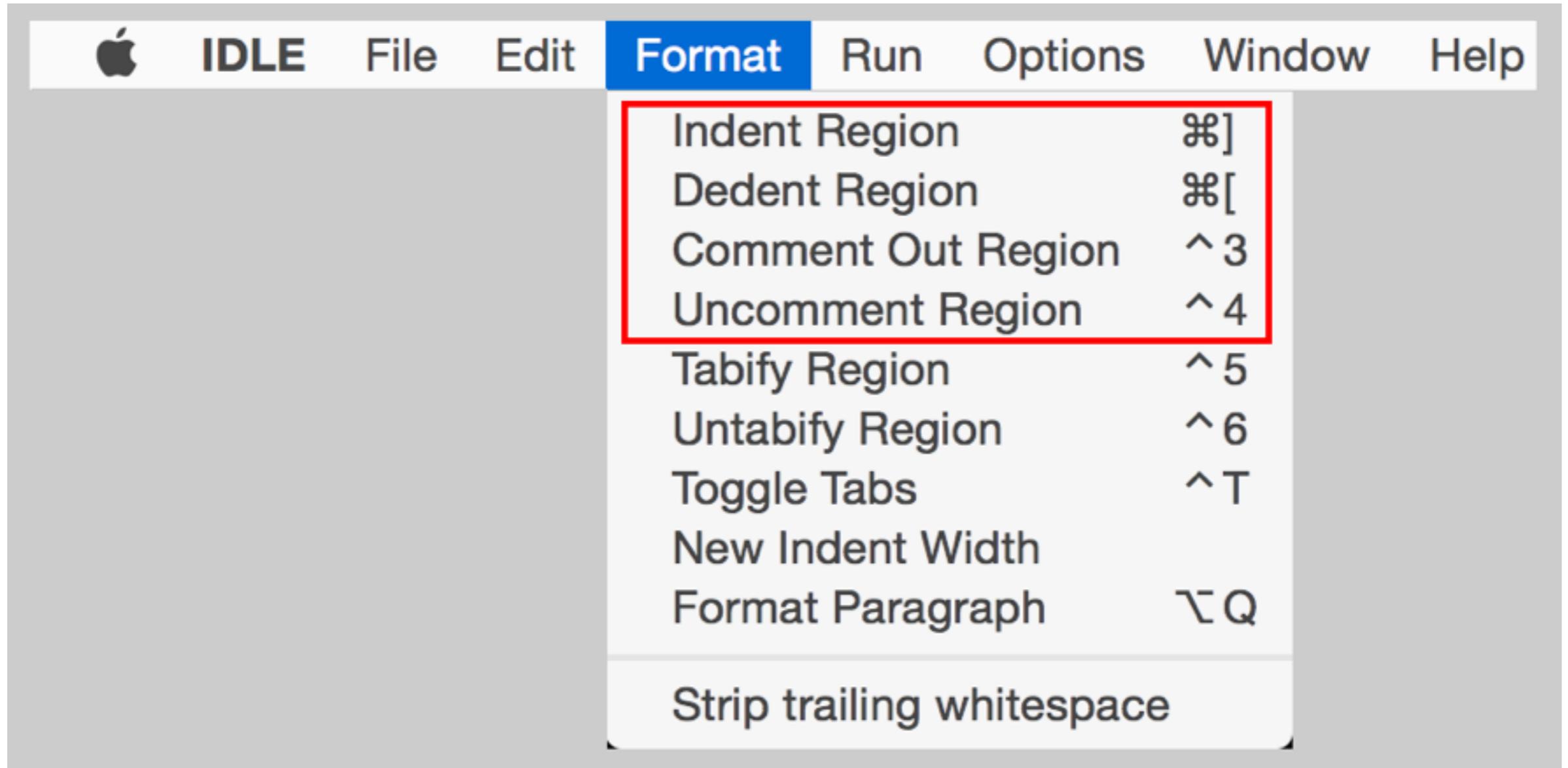


# Stopping a runaway program

- A program error can sometimes lead to an *infinite loop*, that is, code that can never stop
- ```
i = 1
while i < 100:
    print(i, i * i)
```
- The error here is that there is nothing in the loop to change the value of **i**, so it's always less than 100
- To stop a runaway program, go to **Shell > Restart Shell** or enter **control-F6**



# Formatting commands



- All the usual
- Indent and dedent region
- Comment out and uncomment



# Line numbers

---

- In the editing window, the line number that your cursor is on is displayed in the bottom right-hand corner
  - This isn't particularly convenient, but it's what IDLE has
- In editing window, you can also choose:  
**Edit > Go to line**



# Starting the debugger

---

- The debugger allows you to step through your program, one statement at a time, to see what it is doing
- Load the program (with **F5**) but don't automatically run it
- From the **Debug** menu, choose **Debugger**
- Then, enter a function call (such as **main()**)
- **>>>**  
**[DEBUG ON]**  
**>>> main()**



# Running the debugger

```
def is_even(n):  
    """Test if the argument is even"""  
    return n % 2 == 0  
  
def main():  
    for i in range(1, 6):  
        print(i, is_even(i))
```

- Use the Debugger buttons
  - **Step:** Next statement
  - **Over:** Skip over the function call
  - **Out:** Skip out of current function
  - **Go:** Run to the end
  - **Quit:** Quit

Debug Control

Go Step Over Out Quit

Stack  Source  
 Locals  Globals

parity.py:3: is\_even()

'bdb'.run(), line 431: exec(cmd, globals, locals)  
'\_\_main\_\_'.<module>(), line 1: main()  
'\_\_main\_\_'.main(), line 12: print(i, is\_even(i))  
> '\_\_main\_\_'.is\_even(), line 3: return n % 2 == 0

Locals

n 1



# Help

---

- The IDLE **Help** menu provides:
  - **IDLE Help** -- Text only, possibly helpful if you need to figure out what a menu item does
  - **Python Docs** -- Links to a *very* useful online web page
    - **Library Reference** -- All the common methods you are likely to want to use
    - **Language Reference** -- Descriptions of the syntax and semantics of Python itself
  - **Turtle Demo** -- Examples of Python programs that use “Turtle graphics,” which will not be covered in this course



# The End

---

As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.

— Maurice Wilkes discovers debugging, 1949