Reminders

Australian National University

- No labs in Week 1. All labs start in Week 2.
- **Python installation help sessions**:
  - Tues – 3-5pm – Birch Building, Lab 1.08
  - Thurs – 11am-1pm – room N114, CSIT Building (#108)
- Important To-do items:
  - Please fill in the Demographic Information Questionnaire on Wattle
  - Sign up to a lab group
    - via myTimetable: https://mytimetable.anu.edu.au/odd/student
  - Login to STREAMS before your first lab (this is required to set things up for you):
    - https://cs.anu.edu.au/streams/login.php

Read the news forum on Wattle

CECC Class Representatives

Class Student Representation is an important component of the teaching and learning quality assurance and quality improvement processes within the ANU College of Engineering and Computer Science (CECC).

The role of Student Representatives is to provide ongoing constructive feedback on behalf of the student cohort to Course Conveners and to Associate Directors (Education) for continuous improvements to the course.

Roles and responsibilities:

- Act as the official liaison between your peers and convener.
- Be available and proactive in gathering feedback from your classmates.
- Attend regular meetings, and provide reports on course feedback to your course convener
- Close the feedback loop by reporting back to the class the outcomes of your meetings.

Why become a class representative?

- **Ensure students have a voice** to their course convener, lecturer, tutors, and College.
- **Develop skills sought by employers**, including interpersonal, dispute resolution, leadership and communication skills.
- **Become empowered**. Play an active role in determining the direction of your education.
- **Become more aware of issues influencing your University** and current issues in higher education.
- **Course design and delivery.** Help shape the delivery of your current courses, as well as future improvements for following years.

- Note: Class representatives will need to be comfortable with their contact details being made available via Wattle to all students in the class.

- For more information regarding roles and responsibilities, contact:
- ANUSA CECC representatives: sa.cecs@anu.edu.au

Want to be a class representative? Nominate today!

Please nominate yourself to your course convener by end of Week 2, Sem 1, 2024.

Interested? Write to comp1730.convenor@anu.edu.au or talk to us after the lecture.

Australian National University

Variables (part II)

COMP1730 & COMP6730

Reading:

Chapter 2 : Downey, *Think Python*

Chapter 2, Sundnes, *ItSPwP*

OR

https://docs.python.org/3/tutorial/index.html

Every variable has a type

Australian National University

- Variable types in python:
  - Integers (type `int`)
  - Floating-point numbers (type `float`)
  - Text strings (type `str`)
  - Truth or Boolean values (type `bool`)
- Variable types determine what we can do with values (and sometimes what the result is)

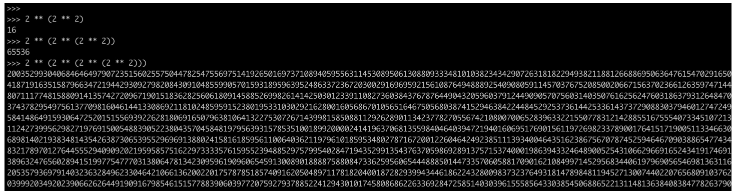
- The `type()` function tells us the type of a variable:

```
python
bash-3.2$
bash-3.2$ python
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> type(2)
<class 'int'>
>>> type(2 / 3)
<class 'float'>
>>> type("zero")
<class 'str'>
>>> type("1")
<class 'str'>
>>> type(1 < 0)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>>
>>>
```

Numeric types: `int`

Australian National University

- `int` types represent the mathematical integers (positive and negative whole numbers) (0, 1, 2, -1, -17, 4096,…)
- Values of type `int` have no inherent size limit in python



- Note: can't use commas to format integers for readability
  - Write `128736` not `1,282,736`

Numeric types: `float`

Australian National University

- Floating-point numbers (type `float`) approximate the mathematical real numbers
- Values of type float have limited range and limited precision
  - Min/max ± 1.79×10^{308}
  - With a few exceptions to this limit
  - Though this is the typical limit – the actual limits depend on the python implementation
- Type `float` also has special values ± `inf` (`float('inf')`, meaning infinity) and `nan` (`float('nan')`, meaning not a number)

String variables

Australian National University

- Strings (type `str`) represent text
- A string literal is enclosed in single or double quote marks

```
>>> "Hello world"
'Hello world'
>>> '4" long'
'4" long'
```

- A string (in python) can contain other types of quote mark, but not the one used to **delimit** it
- More about strings (so much more) in a coming lecture

Type casting: `int()`, `str()`, `float()`

Australian National University

- When a variable is first defined, python will take a guess about type.
- When we want to convert between variable types, or be *explicit* about type when it may not be obvious -- use `int()`, `str()` and `float()`
- There is no automatic type conversion. So, need to convert between types when necessary.

```
>>>
>>> string_var = 'some text'
>>> int_var = 4
>>> float_var = 4.4
>>> str(float_var)
'4.4'
>>> int(float_var)
4
>>> float(str_var)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'str_var' is not defined
>>>
```

Most variables can be changed

Australian National University

- Just in case it comes as a surprise, you are able to re-assign the values of most variables in python. This removes the previous value and replaces it with the new value.
- Most variables in your programs can change – and that is good too

```
>>> x = 5
>>> x
5
>>> x = 7
>>> x
7
```

- Increment and Decrement for counting

```
>>> x = 0
>>> x = x + 1
```

Downey (2015) *Think Python*, 2nd Ed.

Variable assignment

Australian National University

- A variable assignment is written:

`var_name = expression`

- This '=' means something different to this '=='
- When executing an assignment, the interpreter:
  1. Evaulates the right-hand side expression
  2. Associates the left-hand side name with the resulting value

Division means a `float`

Australian National University

- Every constant (literal) with a decimal point represents a `float`:

```
>>> type(1.5 - 0.5)
<class 'float'>
>>> type(1.0)
<class 'float'>
```

- The result of division is always a float:

```
>>> type(4 / 2)
<class 'float'>
```

- Floats can be written (and sometimes printed) in scientific notation:
  - `2.99e8` means $2.99 \cdot 10^{8}$
  - `6.626e-34` means $1.626 \cdot 10^{-34}$
  - `1e308` means $1 \cdot 10^{308}$

String variables are sequences

- Each of the characters in a string may be treated individually. Because `str` variables are **sequences**. More on this in later lectures.
- To access each character in a string, you use the index value (enclosed in square brackets []):

```
>>>
>>> some_text = "Hello, world!"
>>> some_text
'Hello, world!'
>>> some_text[0]
'H'
>>> some_text[5]
','
>>> some_text[7]
'w'
>>>
```

- Index values always start counting from zero!

Lists (quick mention)

- There are other **sequence** variable types in python. These are very useful and the topic of whole later lectures.
- Lists – a sequence of variables that are ordered by index.

```
>>>
>>> first_list = ["0", 1, 2, "three", 4.0]
>>> first_list
['0', 1, 2, 'three', 4.0]
>>> first_list[0]
'0'
>>> first_list[3]
'three'
>>> first_list[4]
4.0
>>>
```

- Lists may contain variables of mixed types, or of a single type.

Dictionaries (quick mention)

- Abbreviated as `dict` variables. Also the topic of a whole lecture, later.
- For storing **key-value pairs** in a single variable. Can use as a **lookup table**. Also very useful:

```
>>>
>>> first_dictionary = {'DOCK2': 'Dedicator of cytokinesis 2', 'CD4': 'Cluster
 of differentiation 4', 'something_else': 123}
>>>
>>> first_dictionary
{'DOCK2': 'Dedicator of cytokinesis 2', 'CD4': 'Cluster of differentiation 4',
 'something_else': 123}
>>>
>>> first_dictionary['DOCK2']
'Dedicator of cytokinesis 2'
>>>
```

- Note the curly braces '{}' for defining a `dict`, and the square brackets '[]' for accessing the values by key.

Variable names

- There are simple rules that govern the names that can be given to variables.
- Good coding practice: make meaningful names that aid understanding
- Names can be long and contain:
  - Uppercase letters
  - Lowercase letters
  - Numbers
  - Underscores _ but not spaces (`this_is_a_variable`, not `this is a variable`)
- Must not start with a number
- Must not contain symbols or illegal characters (apart from underscore)
- Here are some illegal variable names:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

Downey (2015) Think Python, 2nd Ed.

Variable names must not be Python keywords

- Don't try to use these as variable names:

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Downey (2015) Think Python, 2nd Ed.

- This will become natural for you after some practice

Avoid built-in function names too

- Will get to built-in functions in a moment. But these names make bad variable names:
- The problem is that these names will 'work' – but have consequences too
- Since python 3.10, there are *soft keywords* too (`match`, `case`)
- My simple rule – no keyword contains underscores between words ☺ … yet.

Built-in Functions

A	**E**	**L**	**R**
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	**F**	**M**	
ascii()	filter()	map()	**S**
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	**G**	**N**	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	**H**	object()	
callable()	hasattr()	oct()	**T**
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		**P**	**V**
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	**Z**
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		_
	iter()		__import__()

Expressions and Statements

- **Expression**: a combination of values, variables and operators

```
>>> 42
42
>>> n
17
>>> n + 25
42
```

- **Statement:** code that has an effect or makes a change to state

```
>>> n = 17
>>> print(n)
```

Downey (2015) Think Python, 2nd Ed.

Numeric operators in python

Operator	Function
+, –, *, /	Standard arithmetic
**	Power (`x ** n` means x^n)
//	Floor division (`9 // 2` gives `4`)
% (modulus)	Remainder (`9 % 2` gives `1`)

- If you have python already installed, try some of these out (with iPython through a terminal, or with Spyder via the console)

Floor division and modulus

Australian National University

- Floor division is a neat trick in Python. It divides two numbers, discards the remainder and returns an integer value:

```
>>> minutes = 105
>>> hours = minutes // 60
>>> hours
1
```

- To get the remainder, use the modulus '%':

```
>>> remainder = minutes % 60
>>> remainder
45
```

Downey (2015) Think Python, 2nd Ed.

Order of operations - Precedence

Australian National University

- You should know this from high school maths
- The order of precedence of mathematical operators
- PEMDAS: parentheses, exponents, multiplication, division, addition, subtraction
- Matters a whole lot in code – PEMDAS is strictly enforced:
  The result of $6 + 4 / 2^2 - 2 * 10$ is very different to $(6 + 4) / (2^2 - 2) * 10$
- If in doubt, just use parentheses. Some overuse of parentheses is much better than coding a bug that is very hard to track down

Comparison operators

Australian National University

Operator	
<, >, <=, >=	ordering
==	equality
!=	Not equal

- Can compare two values of the same type (for almost any type)
- Comparisons return a *truth* value (Booleans, type `bool`), which is either `True` or `False`
- *Caution*: Conversion from any type to `bool` happens automatically, but the result may not be what you expect.

```
>>> "False"
'False'
>>> bool("False")
True
>>>
```

* *Don't compare `float`s with ==*

String operations

Australian National University

- Funny use of mathematical '+' and '*' operators on strings
- This is a common syntactic shorthand in many languages, but the specifics differ from language to language
- In Python:

```
>>> sentence = 'This' + 'is' + 'a' +'sentence'
>>> print(sentence)
Thisisasentence

>>> 'a' * 3
aaa

>>> ('a' + 'b' + 'c') * 3
abcabcabc
```

Examples:

Australian National University

- Modulus
- PEMDAS
- Comparison operators
- float comparison
- string operations

Comments

Australian National University

- Use them! It is good programming practice
- Sensible use of comments throughout your code is a good habit to cultivate
- Makes your code easier to read and be understood by others
- Will help you remember what you did
- Can start as a structure to guide your coding – like writing pseudocode

- If you ever become part of larger, group-programming projects, your commenting style will really begin to matter.

Comments: The # symbol

Australian National University

- In python, and many other languages, the remainder of a line following a '#' will be ignored by the interpreter/compiler

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

```
percentage = (minute * 100) / 60     # percentage of an hour
```

Downey (2015) Think Python, 2nd Ed.

Multi-line comments """

Australian National University

- Sometimes is it useful to have comments that are a paragraph of text
  - Using # at the beginning of every line can become annoying

- Instead, bound the paragraph with three " symbols together:

```
"""     Here is a multi-line comment that allows
        a block of text
        so I can waffle on about my code design
        or include good usage notes for my script  """
```

- Can be ''' instead (three single quotes)

Types of errors

Australian National University

- Every beginning (and experienced) programmer makes errors!
- Learning to remain calm and know what to do when you see one is part of the programmer skillset. Python is quite chatty
- These can be classified into:
- Syntax errors – the language usage you have written is not understood
- Runtime errors – an exception is raised. Your code is legal, but at runtime something unexpected occurred.
- Semantic errors – Your code runs without error, but does the wrong thing.

Examples:

Australian National University

- Comments:
  - #
  - """
  - '''
- A syntax error
- A runtime error
- A semantic error

Exercises

Australian National University

- Complete Exercises 2-1 and 2-2 of *Think Python*.

Reading

- Chapter 2 of *Think Python* AND/OR
- Chapter 2 of *Introduction to Scientific Programming with Python* AND/OR
- https://docs.python.org/3/tutorial/introduction.html

Installing python with Anaconda

You must try this before labs start next week

Australian National University

Installing python

- Every book about beginning python starts with ‘installing python’
- Some computer OS have python installed by default, others don’t. Some will only have python 2 installed.
- We will solve many difficulties if we all install python with Anaconda

Anaconda?

- The cool thing about python is the number of external code libraries you can use.
- A less-cool thing about python is the job of installing the libraries you want AND all of their dependencies. This is where Anaconda helps out (and mostly makes is easier…). Yes, there is PIP also (and if you like that, cool).
- For now, we will use Anaconda to install a recent release of python and a couple of IDEs

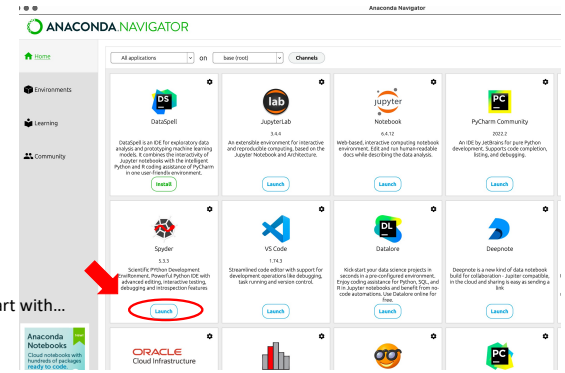
Install instructions:

- The long description is here (with tips and help): https://comp.anu.edu.au/courses/comp1730/labs/install/
- The short description is - go to: https://www.anaconda.com/download
- If you get stuck:
  - **Python installation help sessions**:
    - Tues – 3-4pm – Birch Building, Lab 1.08
    - Thurs – 11am-noon – room N114, CSIT Building (#108)
  - Some help will be available in labs in Week 2, but this will a long first lab, so don’t leave it until then, please.

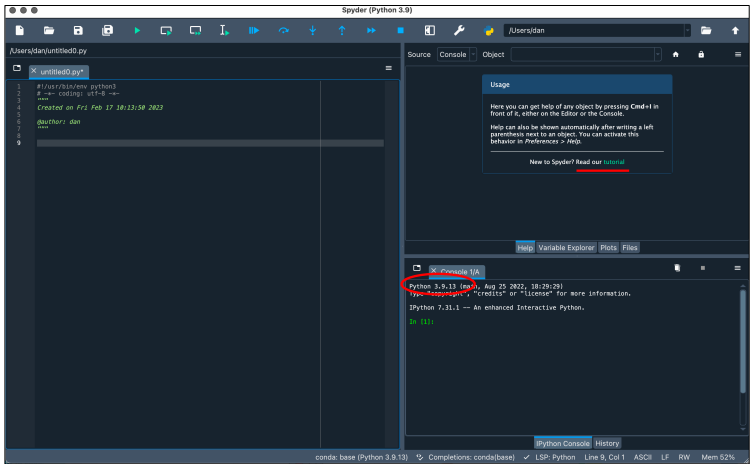
Anaconda comes with many different IDEs:

- Spyder 5.3.3 via Anaconda worked for me, but it is a bit flakey.
- VSCode 1.74.3 worked out of the box

This might a SLOW to start with…



Spyder



Check the Python version you get with Anaconda.

I got 3.9 a month back. Obvs the installer used what was already on my laptop?

Should be as new or newer than this.

Python install – what might go wrong

Australian National University

- Please test your installation – see here:
  - https://comp.anu.edu.au/courses/comp1730/labs/install/
- Let Anaconda do an upgrade when you first start it.
- Maybe you already have python installed?
  - And maybe it is super old, like python 2 or python pre-3.8
- **Python installation help sessions**:
  - Tues – 3-5pm – Birch Building, Lab 1.08
  - Thurs – 11am-1pm – room N114, CSIT Building (#108)
- Some help will be available in labs in Week 2, but this will a long first lab, so don't leave it until then, **please**.