Welcome to the labs!

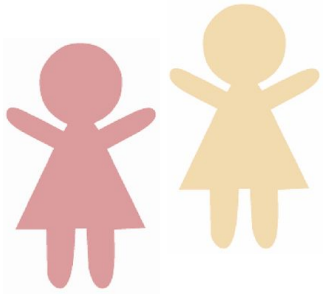
Cryptography

Who are the tutors?

Who are you?

Two Truths and a Lie

1. Get in a group of 3-5 people
2. Tell them three things about yourself:
   a. Two of these things should be true
   b. One of these things should be a lie!
3. The other group members have to guess which is the lie



Log on

Log on and jump on the GPN website

http://bit.ly/gpn-2019-1

You can see:

- These **slides** (to take a look back or go on ahead).
- A digital copy of your **workbook**.
- Help bits of text you can **copy and paste**!

There's also links to places where you can do more programming!

Tell us you're here!

Click on the
Start of Day Survey
and fill it in now!

Today's project!

Cryptography

Using the workbook!

The workbooks will help you put your project together!

Each **Part** of the workbook is made of tasks!

Tasks - The parts of your project

Follow the tasks **in order** to make the project!

Hints - Helpers for your tasks!

Stuck on a task, we might have given you a hint to help you **figure it out**!

The hints have **unrelated** examples, or tips. **Don't copy and paste** in the code, you'll end up with something **CRAZY**!

Task 6.2: Add a blah to your code!

This has instructions on how to do a part of the project

1. **Start by doing this part**
2. **Then you can do this part**

Task 6.1: Make the thing do blah!

Make your project do blah ….

Hint

A clue, an example or some extra information to help you **figure out** the answer.

```
print('This example is not part of the project' )
```

Using the workbook!

The workbooks will help you put your project together!

Check off before you move on from a **Part**! Do some bonuses while you wait!

Checklist - Am I done yet?

Make sure you can tick off every box in this section before you go to the next Part.

Lecture Markers
This tells you you'll find out how to do things for this section during the names lecture.

Bonus Activities
Stuck waiting at a lecture marker?
Try a purple bonus. They add extra functionality to your project along the way.

☑ **CHECKPOINT** ☑

If you can tick all of these off you're ready to move the next part!
☐ Your program does blah
☐ Your program does blob

★ **BONUS 4.3: Do some extra!**

Something to try if you have spare time before the next lecture!

Intro to Caesar Ciphers

Let’s get encoding!

What is a cipher?

A cipher is a way to write a message so that no one else can read it!

Unless they know the secret!

Examples of ciphers

If you've ever made up your own secret language or made notes to your friends so that other people can't read them, you've made a cipher!

For example:

gnidoc evol i

Can you figure out what this says?

It says **I love coding** backwards!

Caesar Cipher

So what's a Caesar Cipher?

It's a cypher that Julius Caesar used in ancient rome to send secret messages to his armies!

Let's learn how it works!

Cipher Wheels

You each have a cipher wheel that looks like this:

You can spin the inside set of letters around and make them line up with different letters

Your Turn!

Now you try on your own!

Try doing the first workbook!

Your tutors are here to help you if you get stuck

Shifting letters

A Caesar Cipher works by shifting letters in the alphabet so that they line up with new letters.

For example if we were to shift everything by 3 it would look like this:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Try turning your purple wheel 3 letters **anti-clockwise** so that you have your letters lining up like this!

Making the secret message

Now, let's write a secret message!

I love coding

For our Caesar cipher we take each letter and replace it with the letter that has been shifted

So, let's start with the letter i

What new letter should we use to replace it?

The letter L

Writing the whole message!

Let's do the rest of the message together

I love coding

l	Is replaced with	o
o	Is replaced with	r
v	Is replaced with	y
e	Is replaced with	h
c	Is replaced with	f
o	Is replaced with	r
d	Is replaced with	g
i	Is replaced with	l
n	Is replaced with	q
g	Is replaced with	j

Secret Message

So our secret encrypted message is

L oryh frglqj

That’s a lot harder to figure out than it just being backwards!

Decrypting

Writing secret messages isn't any fun if you can't figure out what they say!

Luckily you can also use your cipher wheel to *decrypt* a secret message.

How do you think we can do that?

What information do we need to know in order to decrypt a secret message?

It’s the key!

To decrypt a secret message **we need to know** the amount that we shifted the wheel when we encrypted it. That number is called **the key**!

Once we know the key we can just turn our wheel the *other* way (clockwise) to decrypt the message!

Let’s check that it works with: L oryh frglqj

Remember that the key is 3!

Turn it back!

l	Is replaced with	**i**
o	Is replaced with	**l**
r	Is replaced with	**o**
y	Is replaced with	**v**
h	Is replaced with	**e**
f	Is replaced with	**c**
r	Is replaced with	**o**
g	Is replaced with	**d**
l	Is replaced with	**i**
q	Is replaced with	**n**
j	Is replaced with	**g**

Fun fact!

Turning the wheel **backwards**

is the same as

reading your wheel **inside out**!

Intro to Vigenere Ciphers

Caesar Cipher

So now you know what a Caesar Cipher is, let's look at a more complicated cipher!

A Caesar Cipher uses just 1 key to encrypt and decrypt the message, a Vigenere cypher uses a whole word as the key!

The keyword

Let’s see how it uses a whole word by doing an example together!

Let’s use the keyword

pizza

Splitting it into keys

Now we take the keyword and we split it into a bunch of keys!

Each letter of the alphabet equals a different number (a=0, b=1, c=2 etc.)

Now we change our keyword into a bunch of different keys by replacing each letter with its number in the alphabet

p	**i**	**z**	**z**	**a**
15	8	25	25	0

Loop the word

Let’s try encrypting a message with our keyword using a Vigenere cipher now!

I love coding

Each letter in our message will line up with a letter in our keyword and we will keep looping the keyword like this:

i	**l**	**o**	**v**	**e**	**c**	**o**	**d**	**i**	**n**	**g**
p	**i**	**z**	**z**	**a**	**p**	**i**	**z**	**z**	**a**	**p**

Using the numbers

Now we replace each letter of our keyword with the numbers that we worked out before:

i	l	o	v	e	c	o	d	i	n	g
15	8	25	25	0	15	8	25	25	0	15

Next we just shift each letter in our message like we do with a Caesar Cipher but with the key that it lines up with.

What key does the letter C use? 15

Making the secret message

i	Using key: **15**	Is replaced with	x
l	Using key: **8**	Is replaced with	t
o	Using key: **25**	Is replaced with	n
v	Using key: **25**	Is replaced with	u
e	Using key: **0**	Is replaced with	e
c	Using key: **15**	Is replaced with	r
o	Using key: **8**	Is replaced with	w
d	Using key: **25**	Is replaced with	c
i	Using key: **25**	Is replaced with	h
n	Using key: **0**	Is replaced with	n
g	Using key: **15**	Is replaced with	v

Secret Message

So our secret encrypted message is

X tnue rwchnv

To decrypt it you do the same thing as you did to encrypt it: line up they keyword with the message and then, just like with the caesar cipher, you turn the wheel *backwards* to undo the encryption and get the secret message!

Turn it back!

x	Using key: **15**	Is replaced with	i
t	Using key: **8**	Is replaced with	l
n	Using key: **25**	Is replaced with	o
u	Using key: **25**	Is replaced with	v
e	Using key: **0**	Is replaced with	e
r	Using key: **15**	Is replaced with	c
w	Using key: **8**	Is replaced with	o
c	Using key: **25**	Is replaced with	d
h	Using key: **25**	Is replaced with	i
n	Using key: **0**	Is replaced with	n
v	Using key: **15**	Is replaced with	g

Your Turn!

Now you try on your own!

Try doing Part 0 - Part 1 of the second workbook!

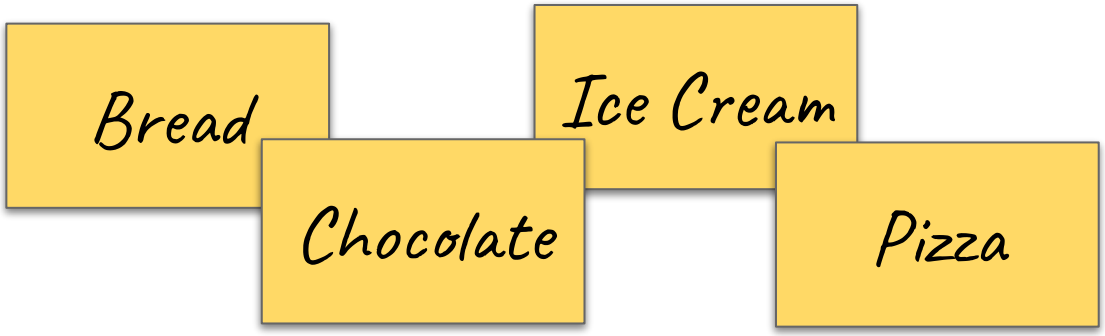
Your tutors are here to help you if you get stuck

Lists

Lists

When we go shopping, we write down what we want to buy!

But we don't store it on lots of little pieces of paper!



We put it in one big shopping list!

- Bread
- Chocolate
- Ice Cream
- Pizza

Lists

It would be annoying to store it separately when we code too!

```
>>> shopping_item1 = "Bread"
>>> shopping_item2 = "Chocolate"
>>> shopping_item3 = "Ice Cream"
>>> shopping_item4 = "Pizza"
```

So much repetition!!

Instead we use a python list!

```
>>> shopping_list = ["Bread", "Chocolate", "Ice Cream",
"Pizza"]
```

You can put (almost) anything into a list

- You can have a list of integers

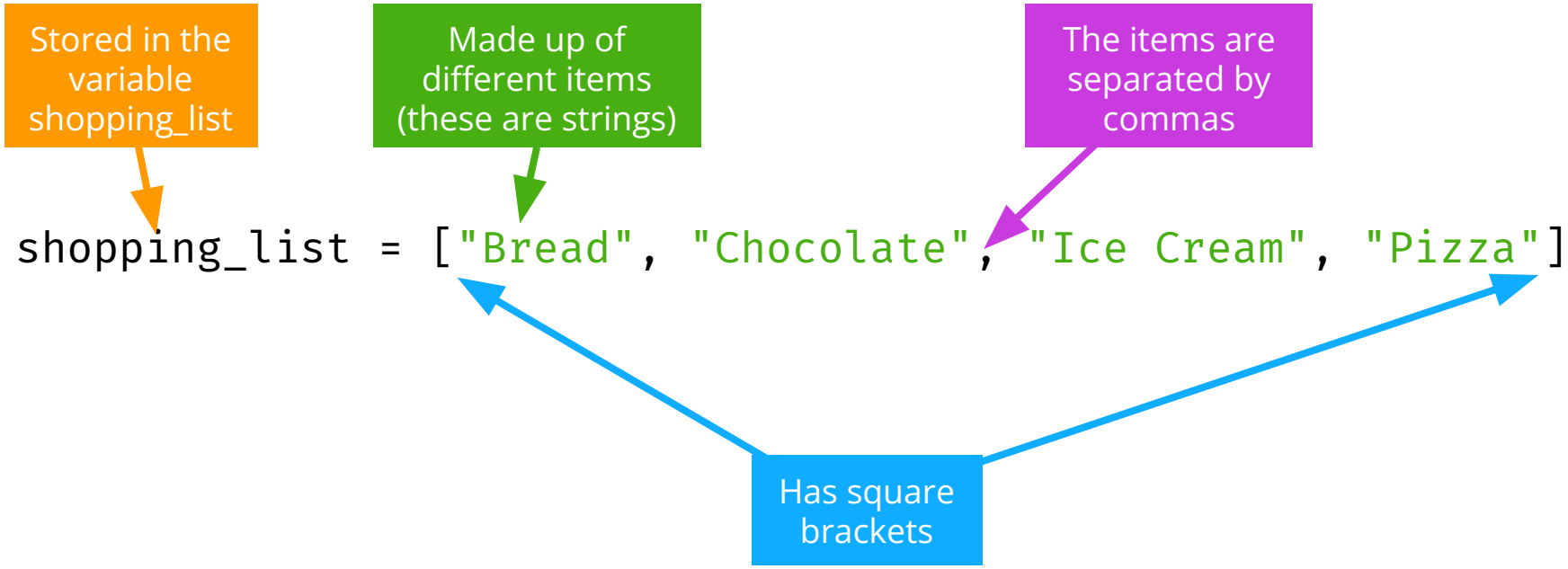
```
>>> primes = [1, 2, 3, 5, 11]
```

- You can have lists with mixed integers and strings

```
>>> mixture = [1, 'two', 3, 4, 'five']
```

- But this is almost never a good idea! You should be able to treat every element of the list the same way.

List anatomy



Try this!

1. Make a list of your favourite things

```
>>> faves = [‘books', ‘butterfly', ‘chocolate', 'skateboard']
```

2. Use `print` to print out your favourite things list

3. Can you make it print on one line?

```
These are a few of my favourite things [‘books', ‘butterfly', ‘chocolate', 'skateboard']
```

>> **Hint: use print with a comma!**

Accessing Lists!

The favourites `list` holds four strings in order.

We can count out the items using index numbers!

0

1

2

3

Remember: Indices start from zero!

Accessing Lists

We access the items in a `list` with an index such as `[0]`:

```
>>> faves[0]
'Books'
```

What code do you need to access the second item in the list?

Going Negative

Negative indices count backwards from the end of the `list`

```
>>> faves = [‘books', ‘butterfly', ‘chocolate',
'skateboard']
>>> faves[-1]
'skateboard'
```

What would `faves[-3]` return?

Falling off the edge

Python complains if you try to go past the end of a `list`

```
>>> faves = ['books', 'butterfly', 'chocolate',
'skateboard']
>>> faves[4]


Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Updating items!

We can also update things in a list:

```
>>> faves = ['books', 'butterfly',
'chocolate', 'skateboard']
>>> faves[1]
'butterfly'
>>> faves[1] = 'kittens'
>>> faves[1]
'kittens'
```

Updating items

What if we decided that we didn't like chocolate anymore, but loved lollipops?




What does this list look like now?




Removing items!

We can remove items from the list if they're no longer needed!

What if we decided that we didn't like butterflies anymore?

```
>>> faves.remove('butterfly')
```

What does this list look like now?

Adding items!

We can also add new items to the list!

What if we decided that we also liked programming?

```
>>> faves.append('programming')
```

What does this list look like now?

List of lists!

You really can put anything in a list, even more lists!

We could use a list of lists to store different sports teams!

```
tennis_pairs = [
    ["Alex", "Emily"], ["Kass", "Annie"], ["Amara", "Viv"]
]
```

Get the first pair in the list

```
>>> first_pair = tennis_pairs[0]
>>> ["Alex", "Emily"]
```

Now we have the first pair handy, we can get the first the first player of the first pair

```
>>> fist_player = first_pair[0]
>>> "Alex"
```

Project time!

You now know all about lists!

Let's put what we learnt into our project.

Try to do Part 2 of the second workbook!

The tutors will be around to help!

Functions!

Simpler, less repetition, easier to read code!

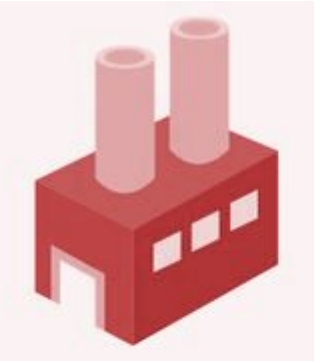
How functions fit together!

Functions are like factories!

Timber Mill

Your main factory!

Metal Worker

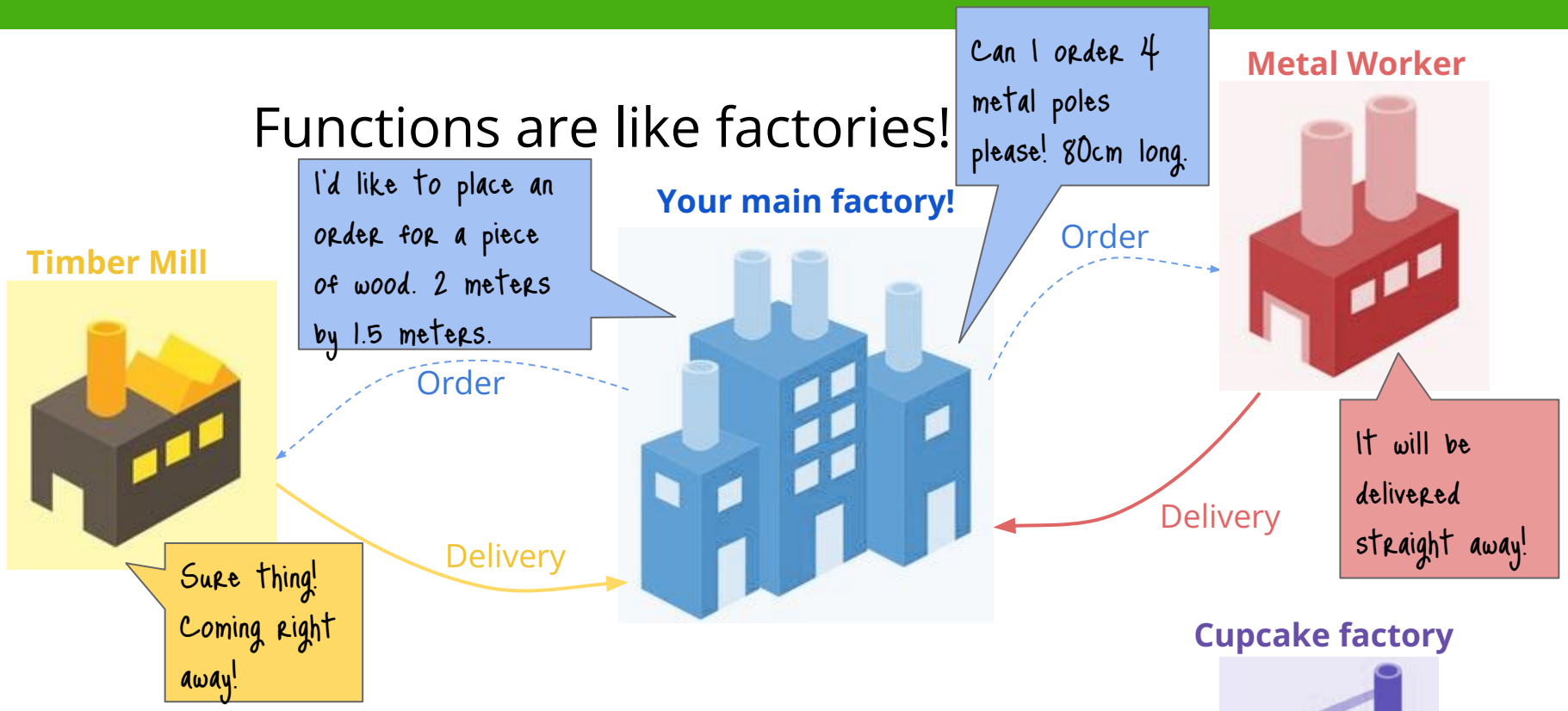


Cupcake factory

Running a factory doesn't mean doing all the work yourself, you can get other factories to help you out!

How functions fit together!

Functions are like factories!

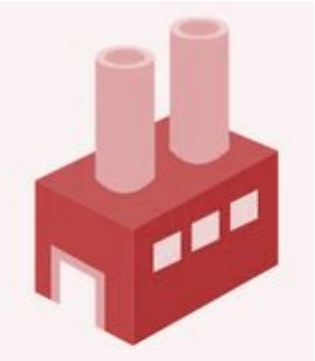


Asking other factories to do some work for you makes your main task simper. You can focus on the assembly!

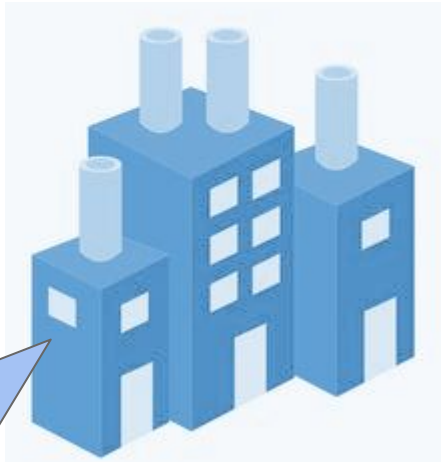
How functions fit together!

Functions are like factories!

Metal Worker



Your main factory!



Timber Mill

Look at this beautiful table I made!

Outsourcing made it simple!

Cupcake factory

How functions fit together!

Your main code!

You can write a bunch of helpful functions to **simplify** your **main goal**!

You can **write** these **once** and then **use** them **lots** of times! They can be **anything** you like!

Helps with printing nicely

Uses stats to make decisions

Does calculations

Don’t reinvent the wheel

We’re already familiar with some python in built functions like print and input!

There’s lots of functions python gives us to save us reinventing the wheel!

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")
11
```

Try these:

```
>>> name = "Renee"
>>> len(name)
5

>>> int("6")
6

>>> str(6)
"6"
```

Defining your own functions

Built in functions are great! But sometimes we want custom functions!

Defining our own functions means:

- We cut down on repeated code
- Nice function names makes our code clear and easy to read
- We can move bulky code out of the way

Defining your own functions

Then you can use your function by calling it!

```
def cat_print():
    print("""
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M """)


cat_print()
cat_print()
```

Which will do this!

```
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
                        #
                         #
                         #
                ^..^ #####
                =TT=      ;
                 #########
                 # #   # #
                 M M   M M
```

Defining your own functions

Then you can use your function by calling it!

```
def cat_print():
    print("""
                    #
                     #
                     #
               ^..^ #####
               =TT=      ;
                #########
                # #   # #
                M M   M M
                   #
                    #
                    #
               ^..^ #####
               =TT=      ;
                #########
                # #   # #
                M M   M M
                   #
                    #
                    #
               ^..^ #####
               =TT=      ;
                #########
                # #   # #
                M M   M M
    """)

cat_print()
cat_print()
```

When using a function in a **script** make sure you define the function first.

It doesn't matter if you call it from inside another function though!

Which will do this!

```
                   #
                    #
                    #
          ^..^ #####
          =TT=      ;
           #########
           # #   # #
           M M   M M
                   #
                    #
                    #
          ^..^ #####
          =TT=      ;
           #########
           # #   # #
           M M   M M
```

Pretty Word Printer

Create a new file and make a pretty word printer! It can print any word you like.

1. Define a function called pretty_word_print

2. Set a variable called word

3. Have the function print out some decorative marks as long as the word above and below the word like these examples:

```
~~~
GPN
~~~
```

```
**********
Hello World
**********
```

4. Call your function in your file as many times as you like!

Functions often need extra information

Functions are more useful if we can change what they do

We can do this by giving them arguments (aka parameters)

```
>>> def hello(person):
...     print('Hello, ' + person + ', how are you?')
>>> hello('Alex')
Hello, Alex, how are you?
```

Here, we give the hello() function a name

Any string will work

```
>>> hello('abcd')
Hello, abcd, how are you?
```

Functions can take multiple arguments

Often we want to work with multiple pieces of information.

You can actually have as many parameters as you like!

This function takes two numbers, adds them together and prints the result.

```
>>> def add(x, y):
...     print(x + y)
>>> add(3, 4)
7
```

Arguments stay inside the function

The arguments are not able to be accessed outside of the function declaration.

```
>>> def hello(person):
...     print('Hello, ' + person + '!')
>>> print(person)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'person' is not defined
```

Variables stay inside the function

Neither are variables declared inside the function. They are **local variables**.

```
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
>>> z
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'z' is not defined
```

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What's the value of z now?

```
>>> print(z)
```

Global variables are not affected

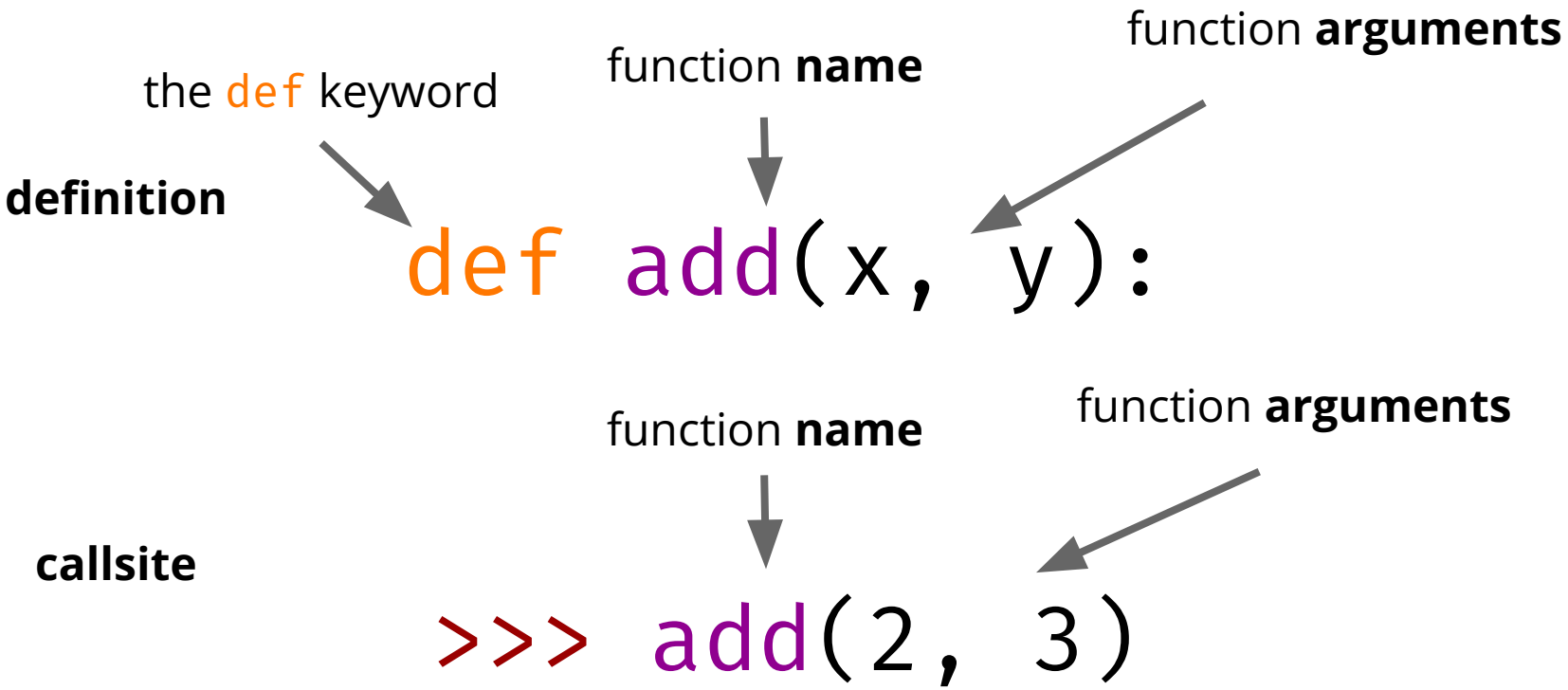
Changing a variable in a function **only changes it *inside* the function.**

```
>>> z = 1
>>> def add(x, y):
...     z = x + y
...     print(z)
>>> add(3, 4)
7
```

What's the value of z now?

```
>>> print(z)
1
```

Recap: A function signature



Pretty Word Printer

At the moment our pretty word printer always prints the same word. Let's fix that!

Edit your pretty word printer function:

1. Change your function so it takes in an argument called word
2. Remove the line where you set word as a variable, now we are passing in word
3. Change the places where you called your pretty word printer, so now you pass in a word as an argument (make sure you pass in a string).
4. Try calling your function multiple times, but with different words

Calling your function with these arguments might look like this:

```
pretty_word_print("Hi everyone")
pretty_word_print("NCSS is cool")
```

```
**********
Hi everyone
**********
**********
NCSS is cool
**********
```

Giving something back

At the moment our function just does a thing, but it's not able to give anything back to the main program.

Currently, we can't use the result of add()

```
>>> def add(x, y):
...     print(x + y)
>>> sum = add(1, 3)
4
>>> sum
```

sum has no value!

Giving something back

Using `return` in a function immediately returns a result.

```
>>> def add(x, y):
...     z = x + y
...     return z
...
>>> sum = add(1, 3)
>>> sum
4
```

Giving something back

When a function returns something, the *control* is passed back to the main program, so no code after the `return` statement is run.

```
>>> def add(x, y):
...     print('before the return')
...     z = x + y
...     return z
...     print('after the return')
>>> sum = add(1, 3)
before the return
>>> sum
4
```

Here, the `print` statement after the `return` never gets run.

First name finder

Make a new file and write a function that takes in a full name (multiple words, separated by spaces) **and returns the first name.**

1. Define a function called get_first_name
2. Make it take 1 argument called full_name
3. Use `full_name.split(" ")` to split the name into a list. Store it in a variable.
4. Return the item at index 0, that's the first name!
5. Call your function, store the returned value and print it out!

Try some different input arguments out! If you do `"Ada Lovelace"` does it return "Ada". What about if you do `"Alan Mathison Turing"`

Project time!

Now you know how to build function!

Now try to do Part 3 - Part 6 of the second workbook!

The tutors will be around to help!

Sets & Files

Sets

Sets are like lists without an order. They're good when you only want to store one of each thing but don't care where they are.

Let's say you want to store your card hand in poker. The order of cards is not important; you only care if a card is in your hand or not! A set lets you look this up quickly.

Sets

1. Create a set

```
>>> hand = set()
```

2. Add to the set

```
>>> hand.add('A hearts')
>>> hand.update(['7 diamonds', 'K clubs'])
>>> hand
{'K clubs', '7 diamonds', 'A hearts'}
```

Sets

3. Check in set

```
>>> if '7 diamonds' in hand:
    print('Play card')
Play card
```

4. Remove from set

```
>>> hand.remove('A hearts')
>>> hand
{'K clubs', '7 diamonds'}
```

Filing it away!

What happens if we want to use different data in our program? What if that data is too big to write in with the keyboard?

We'd have to change our code!!

It would be better if we could keep all our data in a file and just be able to pick and choose what file we wanted to play today!

people.txt

```
Aleisha,brown,black,hat
Brittany,blue,red,glasses
Charlie,green,brown,glasses
Dave,blue,red,glasses
Eve,green,brown,glasses
Frankie,hazel,black,hat
George,brown,black,glasses
Hannah,brown,black,glasses
Isla,brown,brown,none
Jackie,hazel,blonde,hat
Kevin,brown,black,hat
Luka,blue,brown,none
```

Opening files!

To get access to the stuff inside a file in python we need to **open** it!

That doesn't mean clicking on the little icon!

```
my_file = open("test.txt")
```

You'll now be able to read the things in `my_file`

If your file is in the same location as your code you can just use the name!

A missing file causes an error

Here we try to open a file that doesn't exist:

```
f = open('missing.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or
directory: 'missing.txt'
```

You can read a whole file into a string

```
>>> my_file = open('haiku.txt')
>>> my_string = f.read()
>>> my_stirng
'Wanna go outside.\nOh NO!
Help! I got outside!\nLet me
back inside!

>>> print(my_stirng)
Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!
```

haiku.txt

```
Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!
```

You can also read in one line at a time

You can use a for loop to only get 1 line at a time!

```
my_file = open('haiku.txt')
for line in f:
    print(line)
```

```
Wanna go outside.

Oh NO! Help! I got outside!

Let me back inside!
```

Why is there an extra blank line each time?

Chomping off the newline

The newline character is represented by '\n':

```
print('Hello\nWorld')
Hello
World
```

We can remove it from the lines we read with .strip()

```
x = 'abc\n'
x.strip()
'abc'
```

x.strip() is safe as lines without newlines will be unaffected

Reading and stripping!

```
for line in open('haiku.txt'):
    line = line.strip()
    print(line)
```

```
Wanna go outside.
Oh NO! Help! I got outside!
Let me back inside!
```

No extra lines!

Using with!

This is a special trick for opening files!

```
with open("words.txt") as f:
  for line in f:
    print(line.strip())
```

It automatically closes your file for you!

It's good when you are writing files in python!

Project time!

Now you know how to read from files!

Go file your knowledge into Part 7 of the second workbook and the third workbook!

The tutors will be around to help!