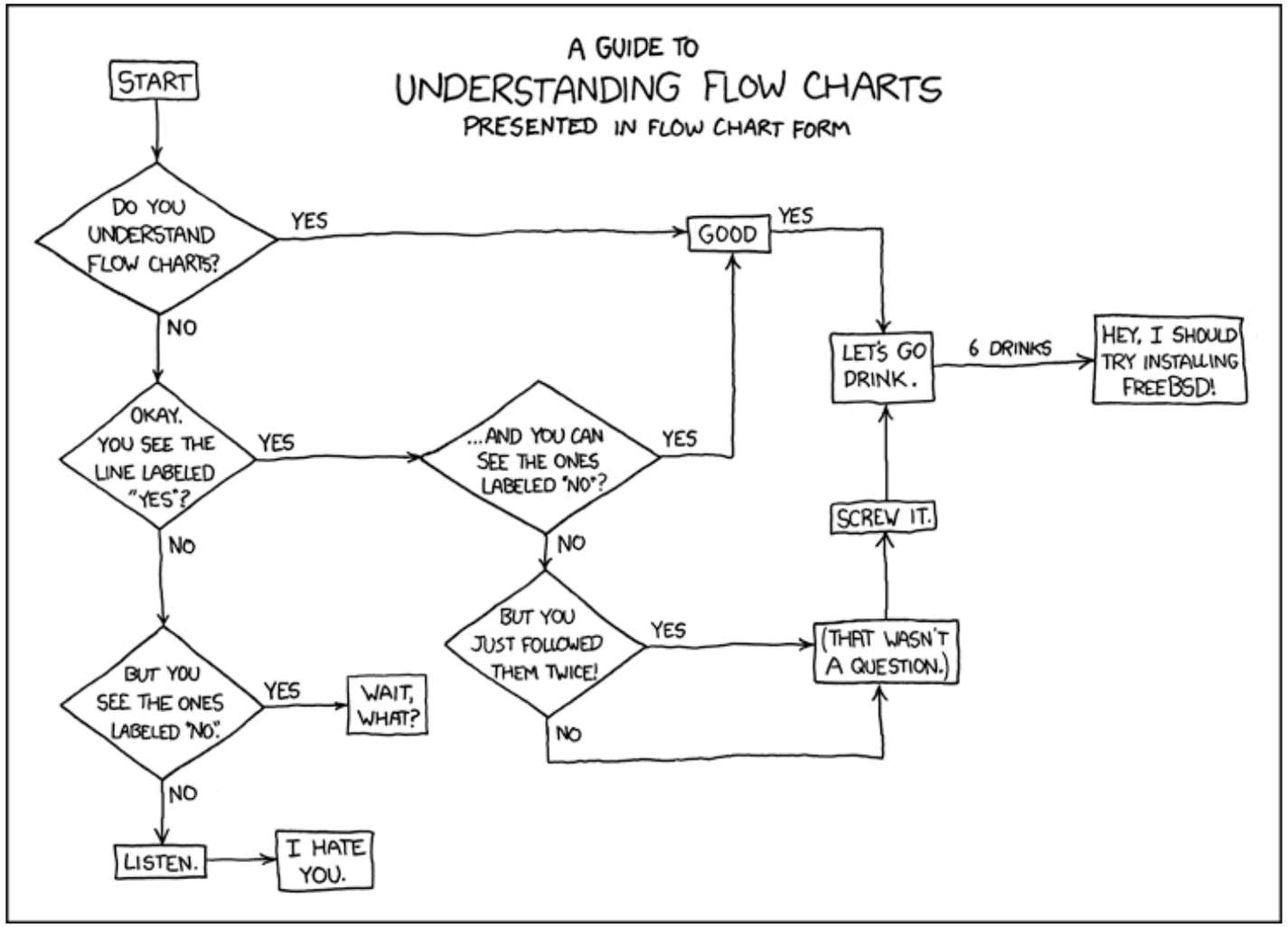
FIT1045: Algorithms and Programming Fundamentals in Python

Lecture 3

Conditional Flow

https://xkcd.com/518/



Don’t forget

Workshops assessment

- You must complete workshop sheet for week 1 and submit online by midnight of **Tuesday 11 August**.
- Make sure to save your work (e.g., Google Drive, Git, USB, laptop)
- Tutor will assess your work during the class.

Tutorial preparation

- You must attempt the tutorial preparation question on your week 2 tutorial sheet by midnight of **Tuesday 11 August**.

Unit email account: fit1045.clayton-x@monash.edu

Objectives of this lecture

Extend Python vocabulary:

- custom functions
- Boolean expressions
- conditional control flow

Covered learning outcomes:

- 1 (translate between problem descriptions and program design with appropriate input/output representations)
- 2 (choose and implement appropriate problem solving strategies in Python)
- 4 (decompose problems into simpler problems and reduce unknown to known problems)

Concrete goal:

Write module for computing customer prices of groceries

Recap

```
>>> x = 1
>>> y = 1.0
>>> z = '1'

>>> username = 'Mario'
>>> 'Hello ' + username
'Hello Mario'

>>> some_ones = max(abs(x - 100), round(0.001)) * z
>>> some_ones
'1111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111'

>>> from math import exp
>>> important_number = exp(x)
>>> important_number
2.718281828459045
```

Recap

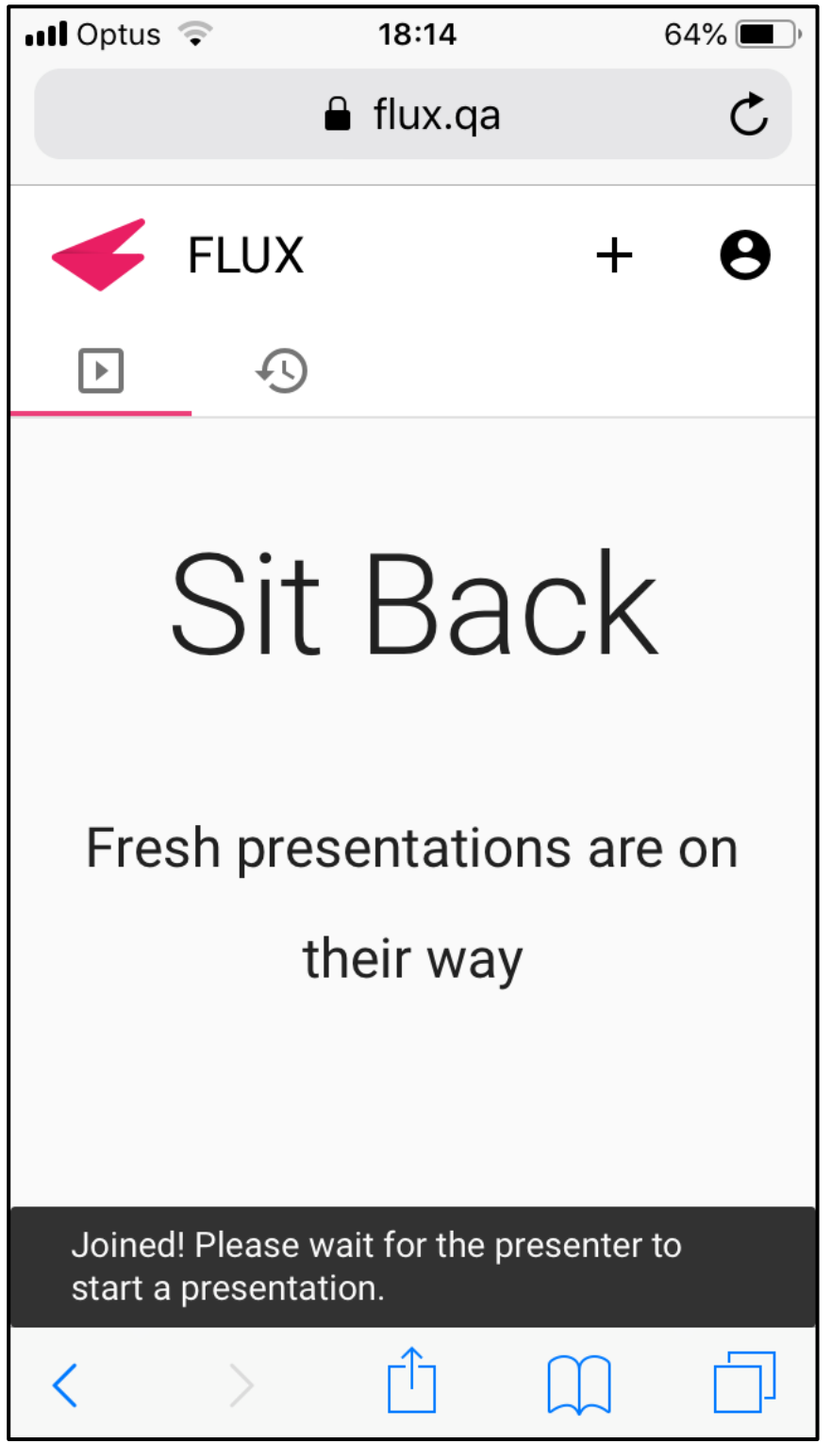
```
>>> help(exp)
```

```
Help on built-in function exp in module math:


exp(x, /)
    Return e raised to the power of x.
(END)
```

Quiz time

1. Visit https://flux.qa
2. Log in (your Authcate details)
3. Touch the + symbol
4. Enter your audience code
   - Clayton: AXXULH
   - Malaysia: LWERDE
5. Answer questions



Where am I?

1. Custom modules and functions
2. Boolean expressions
3. Conditionals

Goal: use Python for grocery price calculation

Our rules for price calculation

- our target sales prices should incorporate a 10% profit margin
- products close to best-before-date should be discounted

  30% reduction rate if within 3 days

  60% reduction rate if within 1 day
- sales prices has to incorporate goods and service tax (GST)

Let's focus first on GST

```
>>> gst_rate = 0.1
>>> gross_price = 35
>>> gst = gross_price * gst_rate
>>> price = gross_price + gst
>>> price
38.5
>>>
```

To be useful we want reusable *function*

```
>>> from prices import price_after_gst
>>> price_after_gst(35)
38.5
>>> price_after_gst(140)
154.0
>>>
```

Too bad Python doesn't come with module 'prices' ☹

… but we can build it ourselves!!!

Start by creating module *file*

Let's put what we have into file: prices.py

```
gst_rate = 0.1
gross_price = 35
gst = gross_price * gst_rate
price = gross_price + gst
```

Next, run Python from same folder

```
Python 3.7.6 (default, Dec 30 2019, 19:38:28)
[Clang 11.0.0 (clang-1100.0.33.16)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import prices
>>> prices.gst_rate
0.1
>>>
```

Now turn specific computation into our own reusable function

```
gst_rate = 0.1


gross_price = 135
gst = gross_price * gst_rate
price = gross_price + gst
```

define this as our function
`price_after_gst`

Step 1: function head

keyword for function definition

function name

colon indicating that following lines contain definition

```
gst_rate = 0.1

def price_after_gst():
gross_price = 135
gst = gross_price * gst_rate
price = gross_price + gst
```

Step 2: function body

```
gst_rate = 0.1

def price_after_gst():
    gross_price = 135
    gst = gross_price * gst_rate
    price = gross_price + gst
```

indentation determines lines of function definition

Must be the same for all lines in block
Python coding standard: always use 4 spaces

Step 3: declare input

```
gst_rate = 0.1

def price_after_gst(gross_price):
    gst = gross_price * gst_rate
    price = gross_price + gst
```

input parameter list
in parentheses

available as
local variable (name)
in function body

Step 4: declare output

```
gst_rate = 0.1

def price_after_gst(gross_price):
    gst = gross_price * gst_rate
    price = gross_price + gst
    return price
```

return keyword followed by result expression

Start function with a string to document it

```
gst_rate = 0.1

def price_after_gst(gross_price):
    """
    Input : gross price of product
    Output: sales price of product (incorporating GST)
    """
    gst = gross_price * gst_rate
    price = gross_price + gst
    return price
```

this is called a "docstring"

Good practice:

write a docstring *before* starting to work on function body (*clarify* what exactly function is supposed to do)

To work with updated module we have to reload it or whole shell

```
def price_after_gst(gross_price):
    """
    Input : gross price of product
    Output: sales price of product (incorporating GST)
    """
    gst = gross_price * gst_rate
    price = gross_price + gst
    return price
```

```
Python 3.7.6 (default, Dec 30 2019, 19:38:28)
[Clang 11.0.0 (clang-1100.0.33.16)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from importlib import reload
>>> reload(prices)
<module 'gst' from '/Users/mbol0005/Google Drive
Monash/FIT1045/FIT1045-S1-2020/Lectures/Lecture03/prices.py'>
>>> from prices import price_after_gst
>>>
```

Our function documentation shows up in help page

```
def price_after_gst(gross_price):
    """
    Input : gross price of product
    Output: sales price of product (incorporating GST)
    """
    gst = gross_price * gst_rate
    price = gross_price + gst
    return price
```

```
Python 3.7.6 (default, Dec 30 2019, 19:38:28)
[Clang 11.0.0 (clang-1100.0.33.16)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from importlib import reload
>>> reload(prices)
<module 'gst' from '/Users/mbol0005/Google Drive
Monash/FIT1045/FIT1045-S1-2020/Lectures/Lecture03/prices.py'>
>>> from prices import price_after_gst
>>> help(price_after_gst)
```

```
Help on function price_after_gst in module gst:

price_after_gst(gross_price)
    Input : gross price of product
    Output: sales price of product (incorporating GST)
```

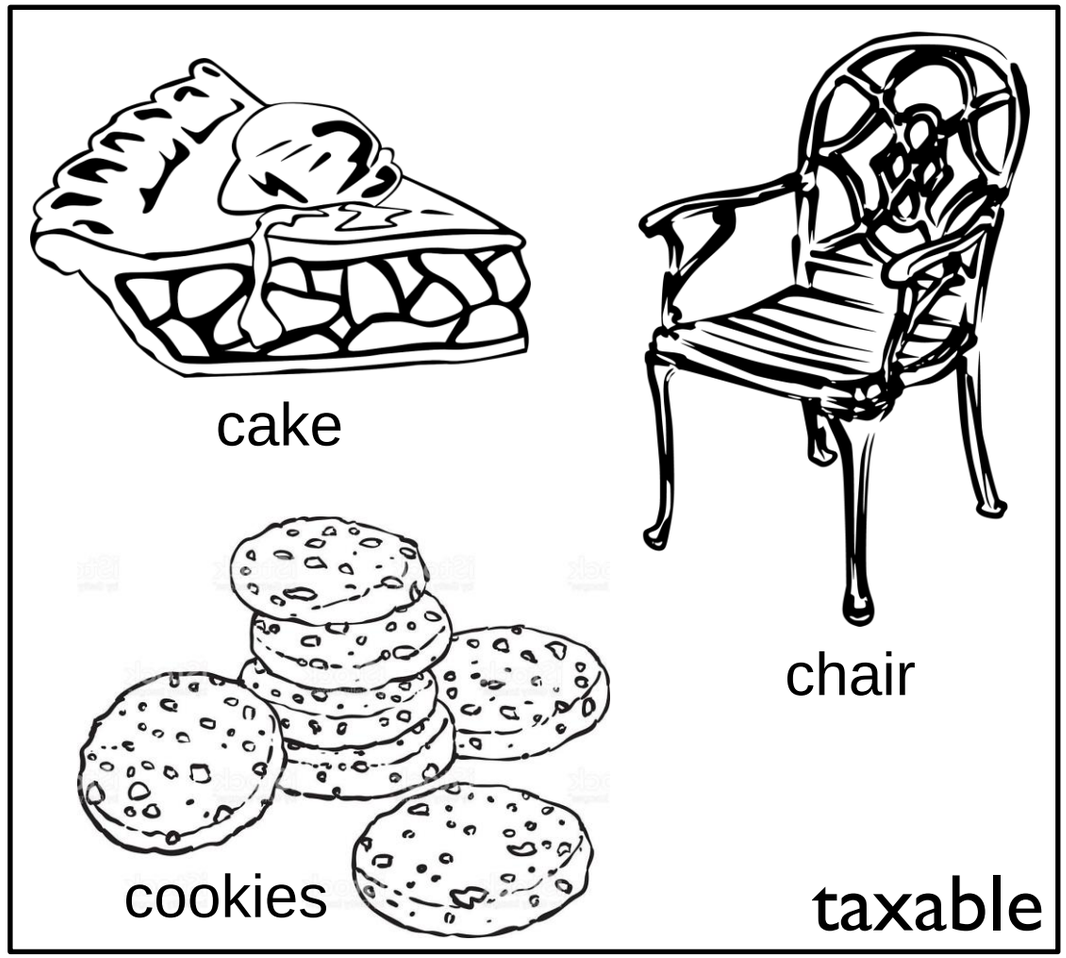
We've done it!!!

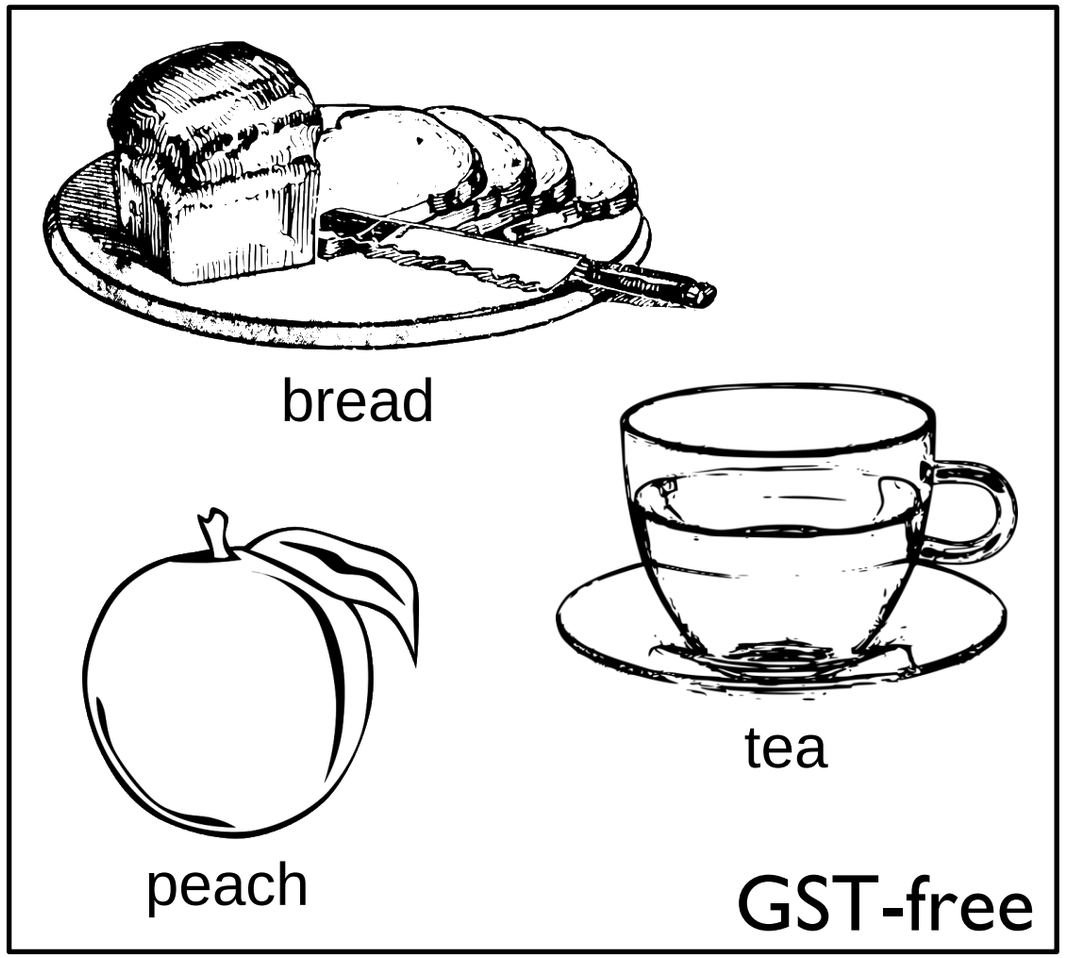
```
gst_rate = 0.1

def price_after_gst(gross_price):
    gst = gross_price * gst_rate
    price = gross_price + gst
    return price
```

```
Python 3.7.6 (default, Dec 30 2019, 19:38:28)
[Clang 11.0.0 (clang-1100.0.33.16)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from importlib import reload
>>> reload(gst)
<module 'gst' from '/Users/mbol0005/Google Drive
Monash/FIT1045/FIT1045-S1-2020/Lectures/Lecture03/gst.py'>
>>> from gst import customer_price
>>> price_after_gst(35)
38.5
>>> price_after_gst(140)
154
```

…or have we?

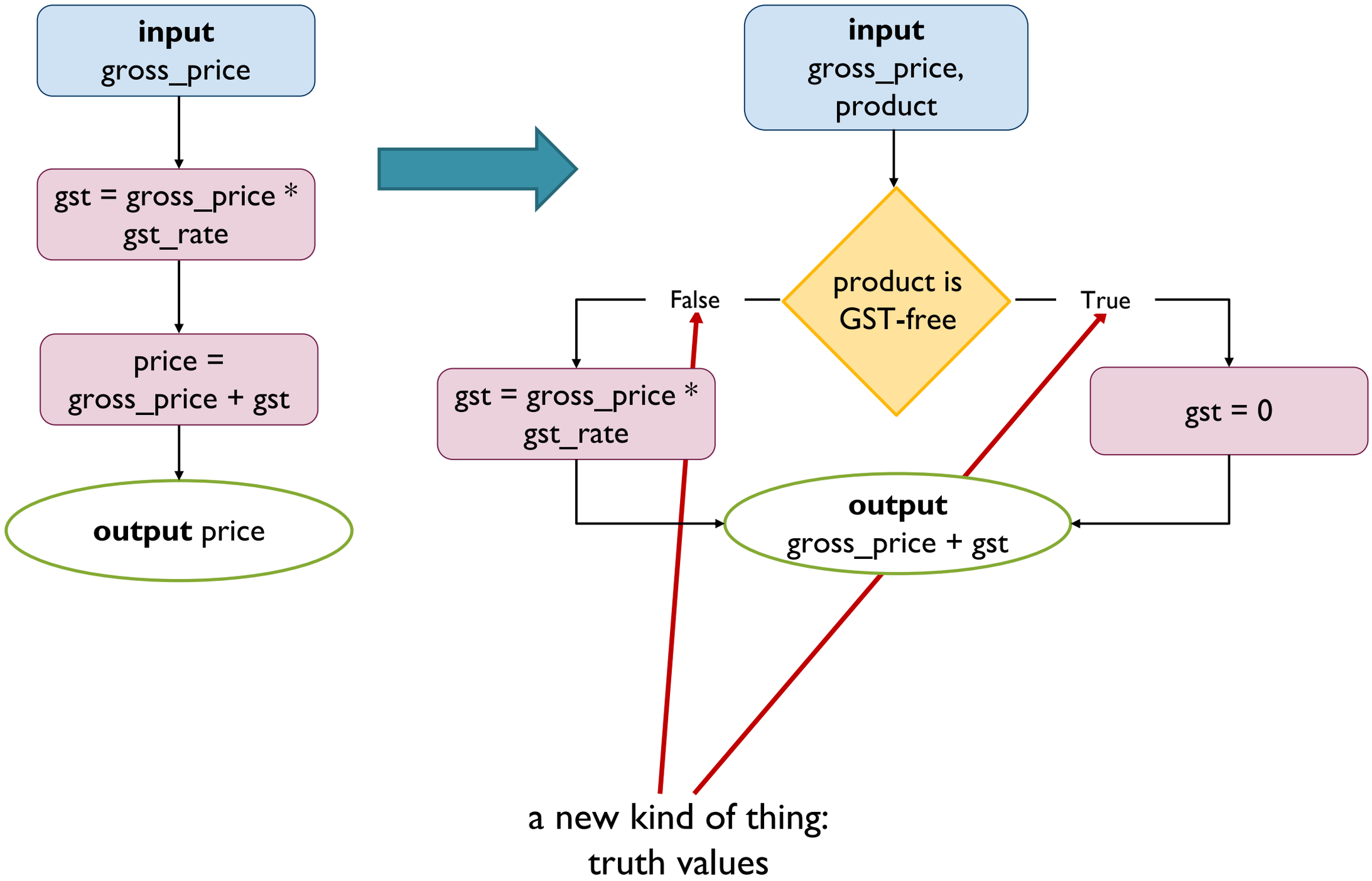




So customer price function has to be modified…

```
>>> price_after_gst(5, 'cookies')
5.05
>>> price_after_gst(5, 'bread')
5
>>>
```

Now we need *conditional* behaviour



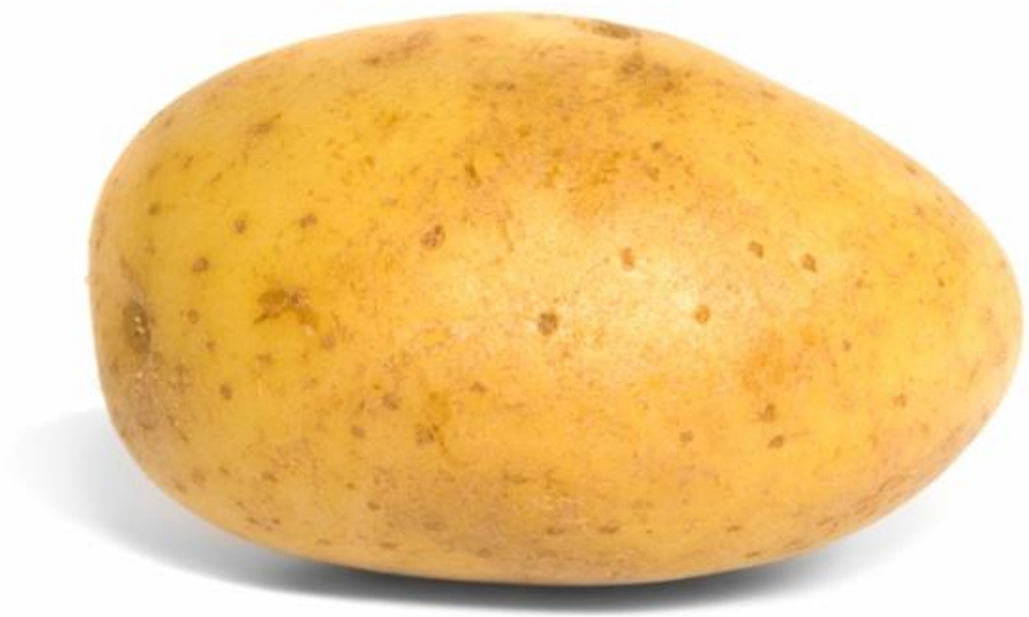
Where am I?

1. Custom functions and modules
2. Boolean values and logic
3. Conditional behaviour

Logic: Reasoning about the Truth

For every thing in the world, it is either a potato, or not a potato.

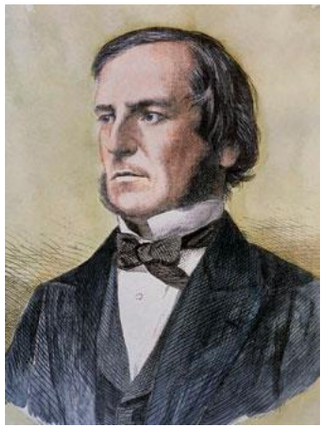
- True
- False



https://flux.qa

Clayton : AXXULH
Malaysia: LWERDE

Boolean values



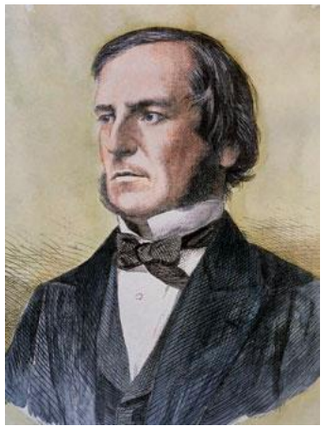
George Boole
1815-64, England

```
>>> True
True
>>> False
False
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>>
```

True and False are the only two objects of type bool

Comparison operators yield Boolean results

Operator	Description
==	Equal
!=	Not equal



George Boole
1815-64, England

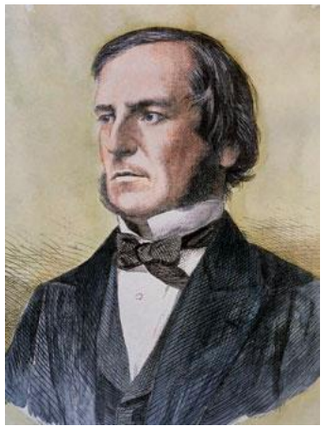
```
>>> 'potato' == 'potato'
True
>>> 'icecream' == 'potato'
False
>>> 'icecream' != 'potato'
True
>>> 1 == 1.0
True
```

note difference to assignment statement

numeric objects of different type can be equal

Don’t confuse equals operator with assignment statement

Operator	Description
==	Equal
!=	Not equal



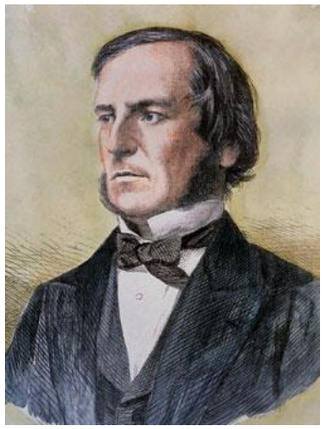
George Boole
1815-64, England

```
>>> thing = 'potato'
>>> thing
'potato'
>>> thing == 'icecream'
False
>>> thing = 'icecream'
>>> thing
'icecream'
>>>
```

very different outcomes

More operators for *ordered comparison*

Operator	Description
==	Equal
!=	Not equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal



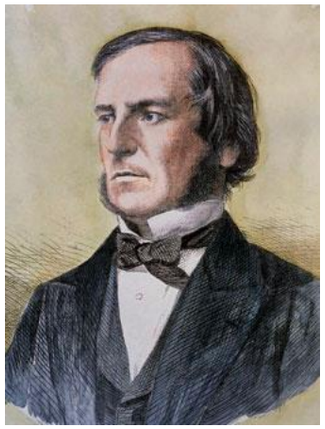
George Boole
1815-64, England

```
>>> 5 < 7
True
>>> 5 > 7
False
>>> -1 >= -1.1
True
>>>
```

again works across numeric types as expected

More operators for *ordered comparison*

Operator	Description
==	Equal
!=	Not equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal



George Boole
1815-64, England

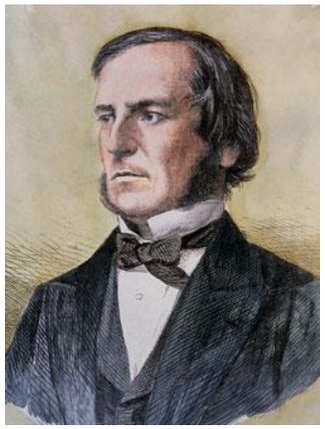
```
>>> 'williams' < 'wilson'
True
>>> 'byron' < 'abrams'
False
>>> [1, 2, 3] < [1, 2, 4]
True
```

for strings and other sequence types comparison uses *lexicographic* order

More operators for *ordered comparison*

Operator	Description
==	Equal
!=	Not equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

https://docs.python.org/3/library/stdtypes.html#comparisons



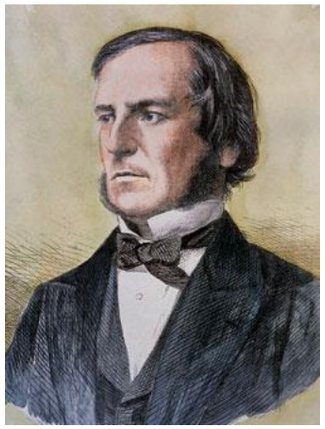
George Boole
1815-64, England

```
>>> 'williams' < 'wilson'
True
>>> 'byron' < 'abrams'
False
>>> [1, 2, 3] < [1, 2, 4]
True
>>> 10 > 'byron'
TypeError: '>' not supported between instances of 'int'
and 'str'
```

Only defined objects that are *comparable* (ordered)

Membership test operator for sequences

Operator	Description
e in x	True if e is a member of x (e is contained in x); False otherwise
e not in x	False if e is a member of x (e is contained in x); True otherwise

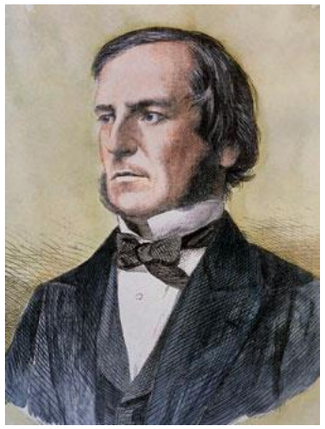


George Boole
1815-64, England

```
>>> 'l' in 'williams'
True
>>> 'l' in 'byron'
False
>>> 2 in [1, 2, 4]
True
>>>
```

Membership test operator for sequences

Operator	Description
e in x	True if e is a member of x (e is contained in x); False otherwise
e not in x	False if e is a member of x (e is contained in x); True otherwise



George Boole
1815-64, England

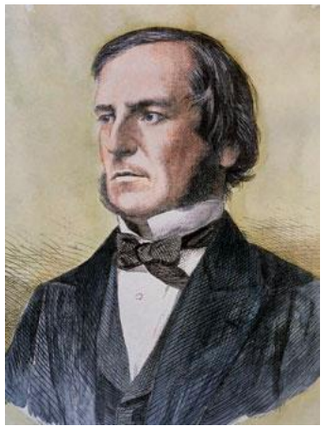
```
>>> 'l' in 'williams'
True
>>> 'l' in 'byron'
False
>>> 2 in [1, 2, 4]
True
>>> "FIT1045" in "I'm studying FIT1045 in S1-2020"
True
>>> [1, 2] in [1, 2, 4]
False
```

Strings define containment as "is sub-string"

Lists define containment strictly as "is element of"

Boolean operators for logical expressions

Operator	Description (simplified)
x or y	True if either x or y are True; otherwise False
x and y	True if both x and y are True; otherwise False
not x	True if *x* is False; otherwise False

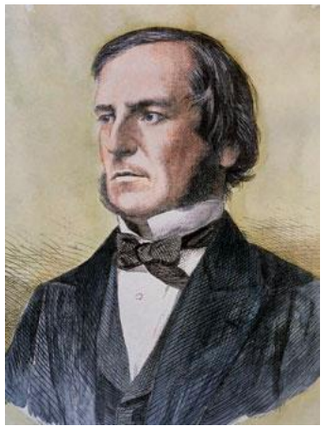


George Boole
1815-64, England

```
>>> True and False
False
>>> True or False
True
>>> not True
False
>>> value = 10
>>> 1 <= value and value <= 100
True
>>>
```

Boolean operators for logical expressions

Operator	Description (simplified)
x or y	True if either x or y are True; otherwise False
x and y	True if both x and y are True; otherwise False
not x	True if *x* is False; otherwise False



George Boole
1815-64, England

```
>>> def is_potato_or_not_potato(thing):
...     return thing == 'potato' or thing != 'potato'
...
>>> is_potato_or_not_potato('potato')
True
>>> is_potato_or_not_potato('icecream')
True
>>> is_potato_or_not_potato(47)
True
```

Boolean operators: Precedence

Operation	Description	Precedence
x or y	if *x* is false, then *y*, else *x*	Lowest
x and y	if *x* is false, then *x*, else *y*	Medium
not x	if *x* is false, then True, else False	Highest

Note: all comparison operators have same precedence as **not**

https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not

not (False and True) or not True and True

not (False and True) or not True and True

not False or not True and True

True or False and True

True or False

True

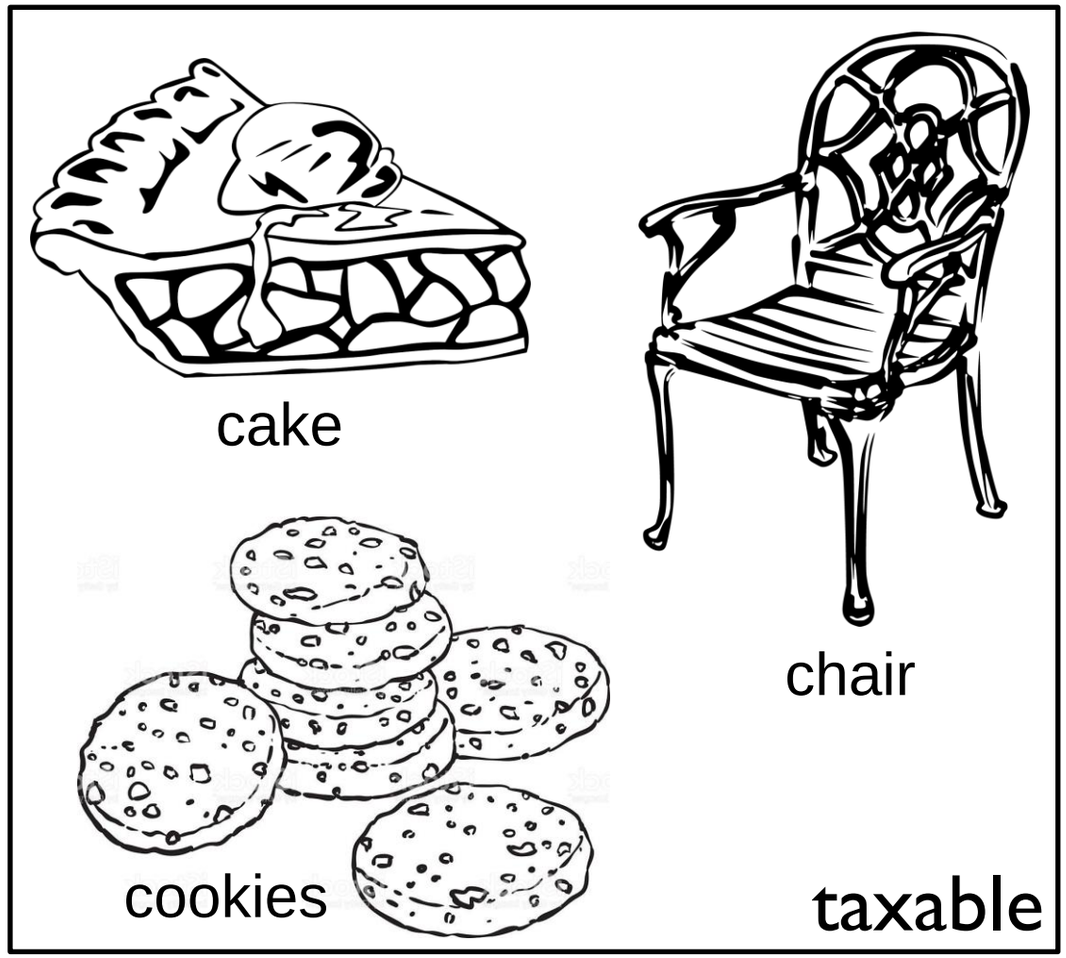
https://flux.qa

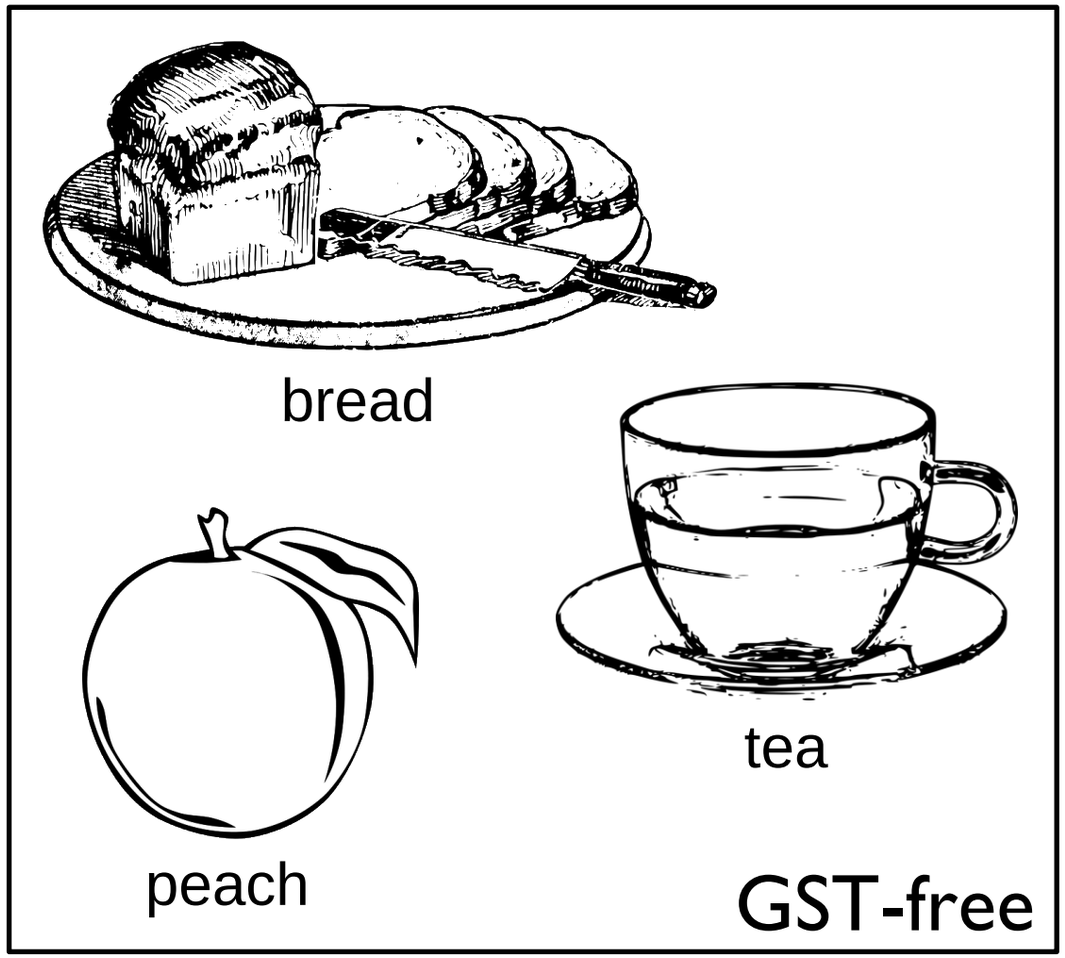
Clayton : AXXULH
Malaysia: LWERDE

Where am I?

1. Recap and custom functions
2. Boolean expressions
3. Conditionals

Back to our GST problem…

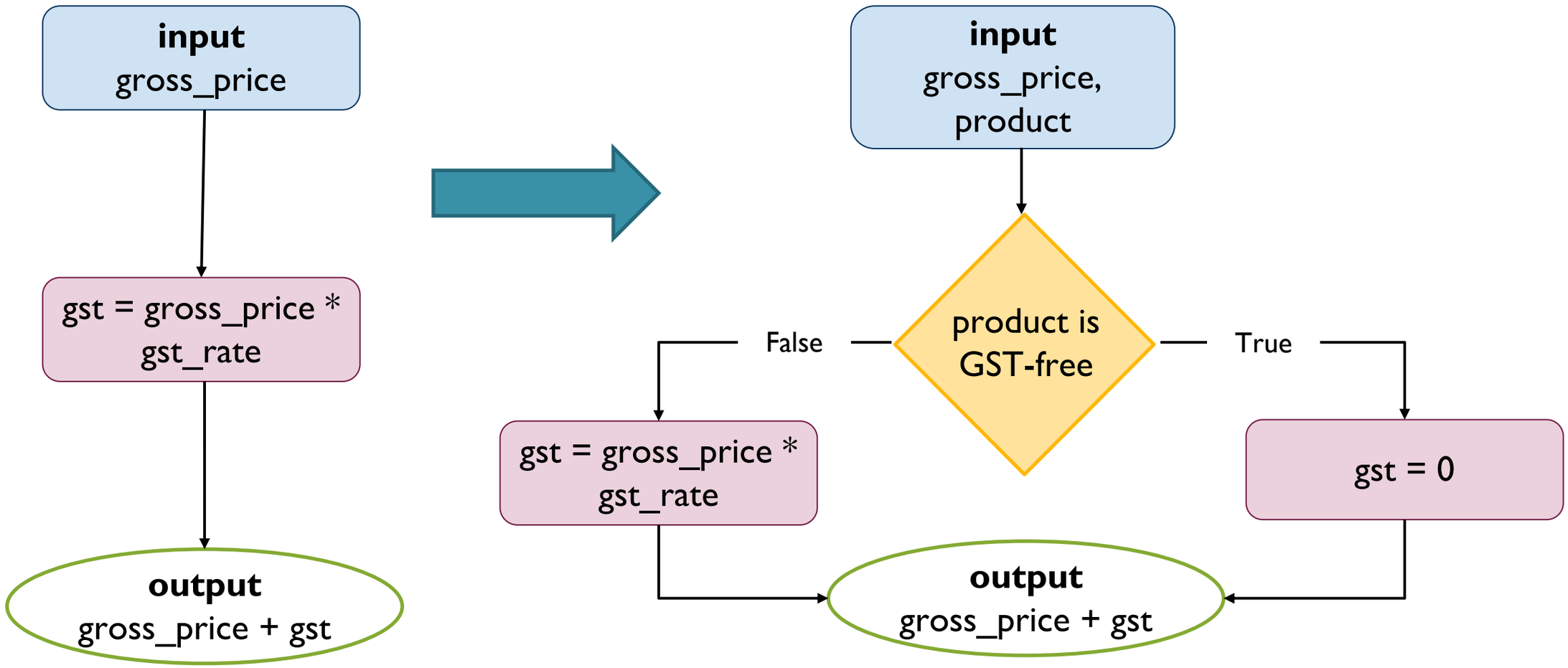




So customer price function has to be modified…

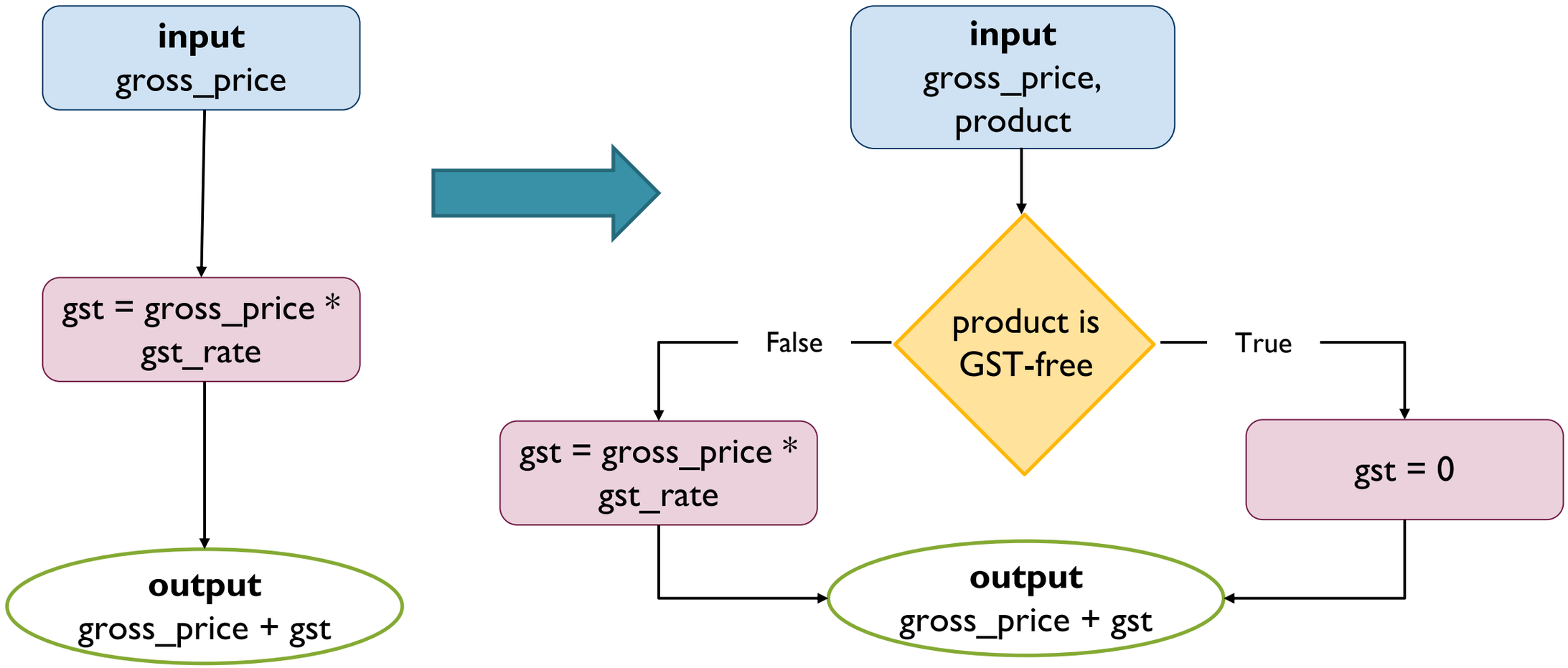
```
>>> price_after_gst(5, 'cookies')
5.05
>>> price_after_gst(5, 'bread')
5
>>>
```

Change from linear to *conditional* flow



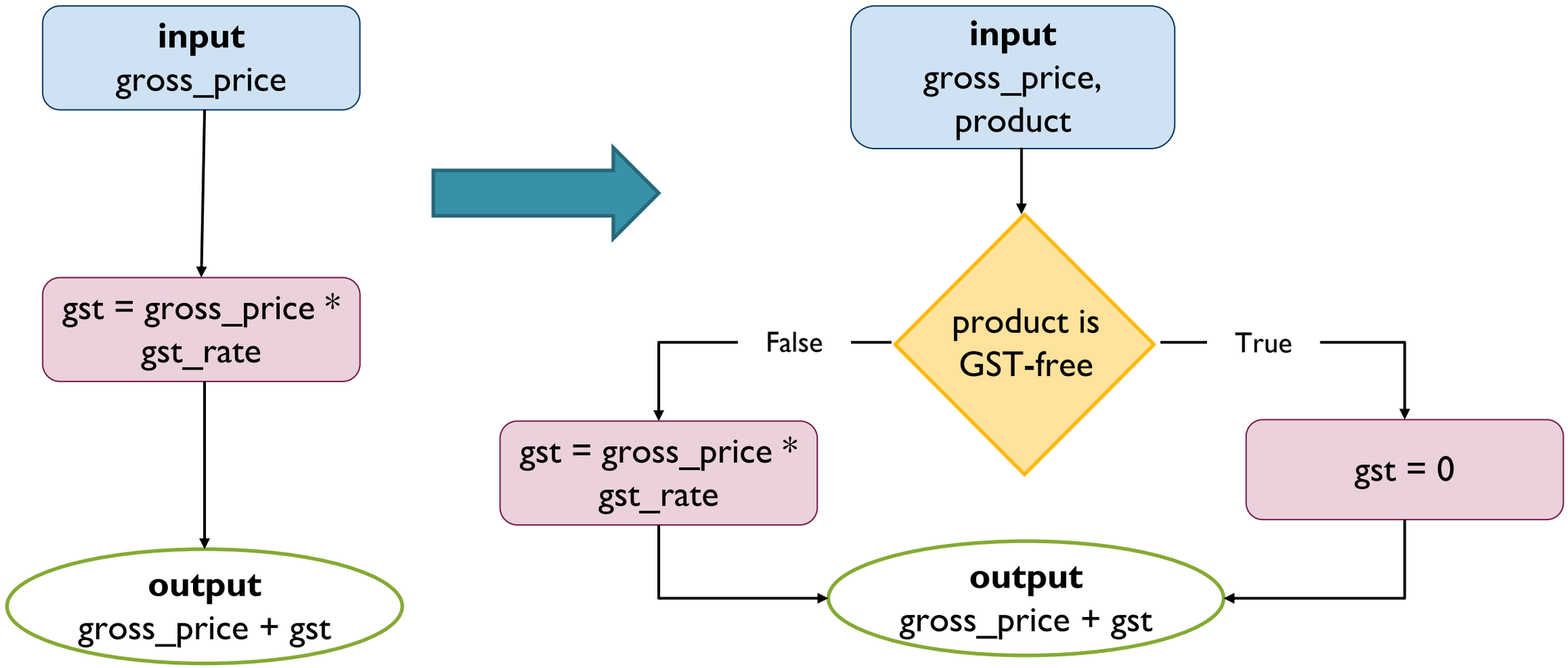

```
def price_after_gst(gross_price):
    gst = gross_price * gst_rate
    return gross_price + gst
```

Extend function *parameter list*




```
def price_after_gst(gross_price, product):
    gst = gross_price * gst_rate
    return gross_price + gst
```

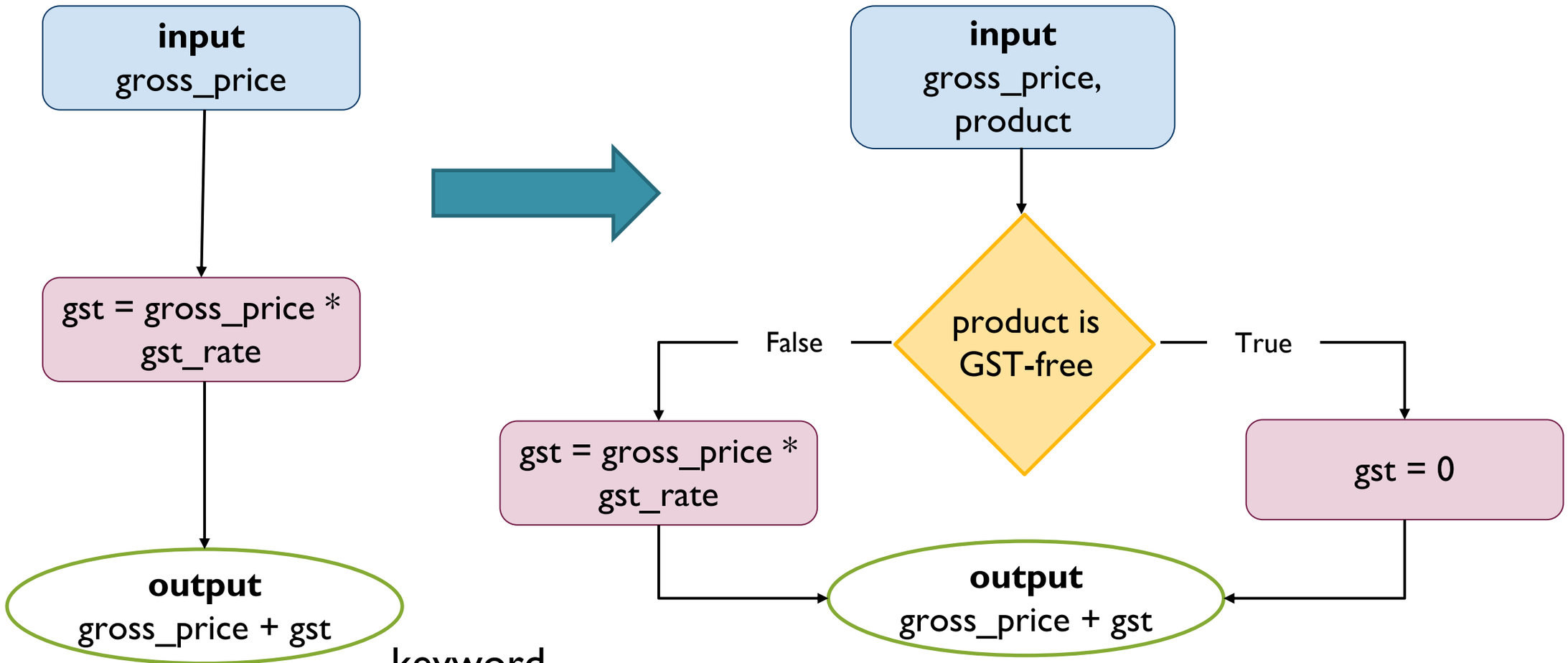
Declare what are GST free product




```
gst_free_products = ['bread', 'peach', 'tea']

def price_after_gst(gross_price, product):
    gst = gross_price * gst_rate
    return gross_price + gst
```

Use **if**-statement for conditional flow



keyword

Boolean expression

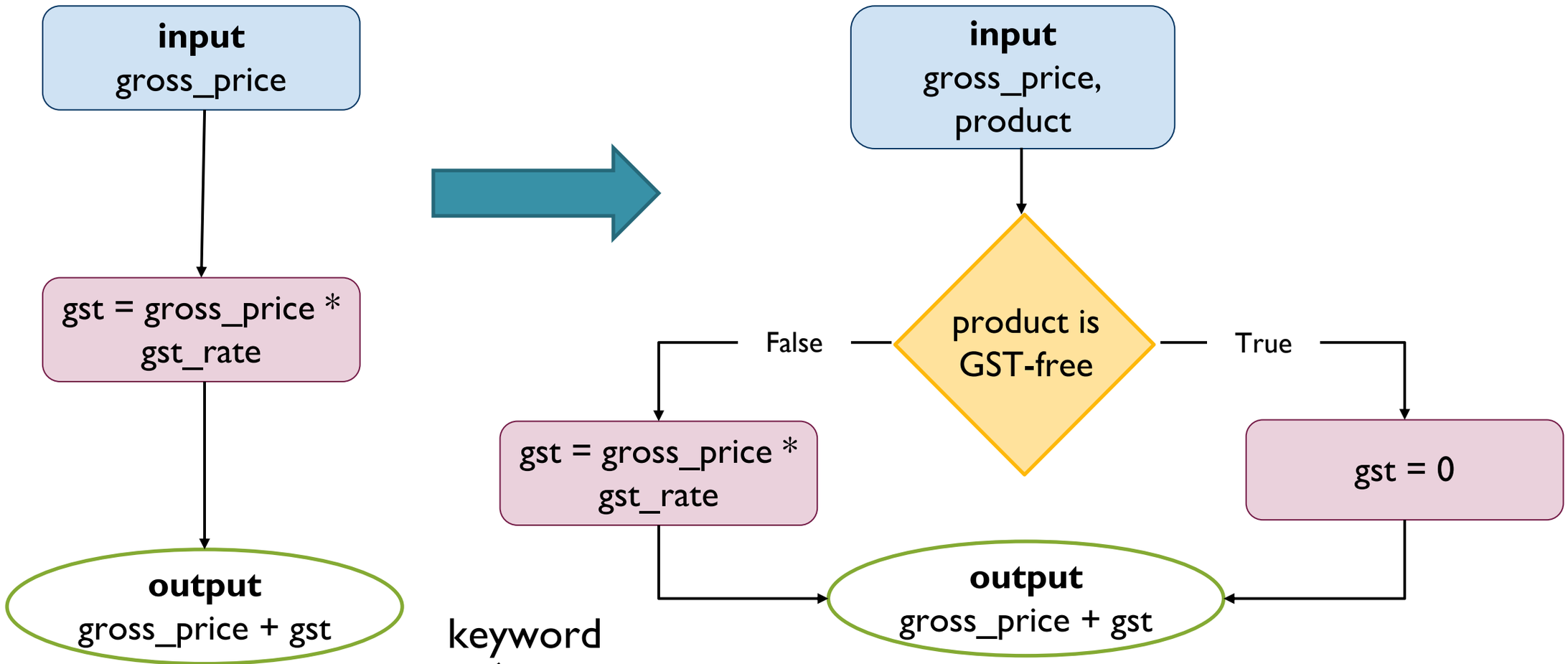
```
gst_free_products = ['bread', 'peach', 'tea']

def price_after_gst(gross_price, product):
    if product in gst_free_products:
        gst = 0

    gst = gross_price * gst_rate
    return gross_price + gst
```

indentation determines conditional code block

Use **else**-block for alternative behaviour



keyword

```
gst_free_products = ['bread', 'peach', 'tea']

def price_after_gst(gross_price, product):
    if product in gst_free_products:
        gst = 0
    else:
        gst = gross_price * gst_rate
    return gross_price + gst
```

Use **else**-block for alternative behaviour

```
>>> reload(prices)
<module 'gst' from '/Users/mbol0005/Google Drive
Monash/FIT1045/FIT1045-S1-2020/Lectures/Lecture03/prices.py'>
>>> from prices import price_after_gst
>>> price_after_gst(5, 'cookies')
5.05
>>> price_after_gst(10, 'icecream')
10.10
>>> price_after_gst(5, 'bread')
5
>>>
```

```
gst_free_products = ['bread', 'peach', 'tea']

def price_after_gst(gross_price, product):
    if product in gst_free_products:
        gst = 0
    else:
        gst = gross_price * gst_rate
    return gross_price + gst
```

Can now use GST function in solution for overall problem

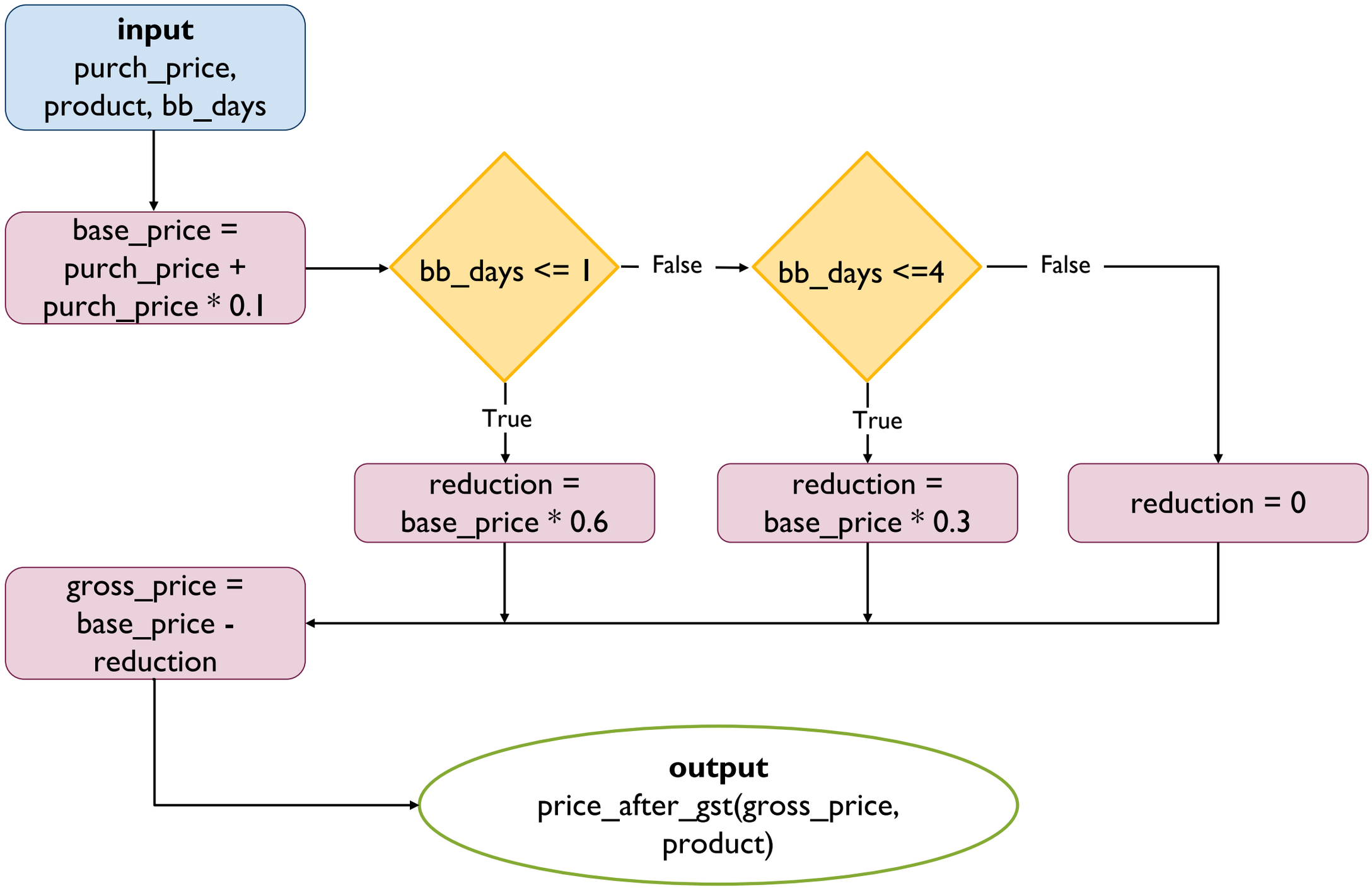
Our rules for price calculation

- our target sales prices should incorporate a 10% profit margin
- products close to best-before-date should be discounted

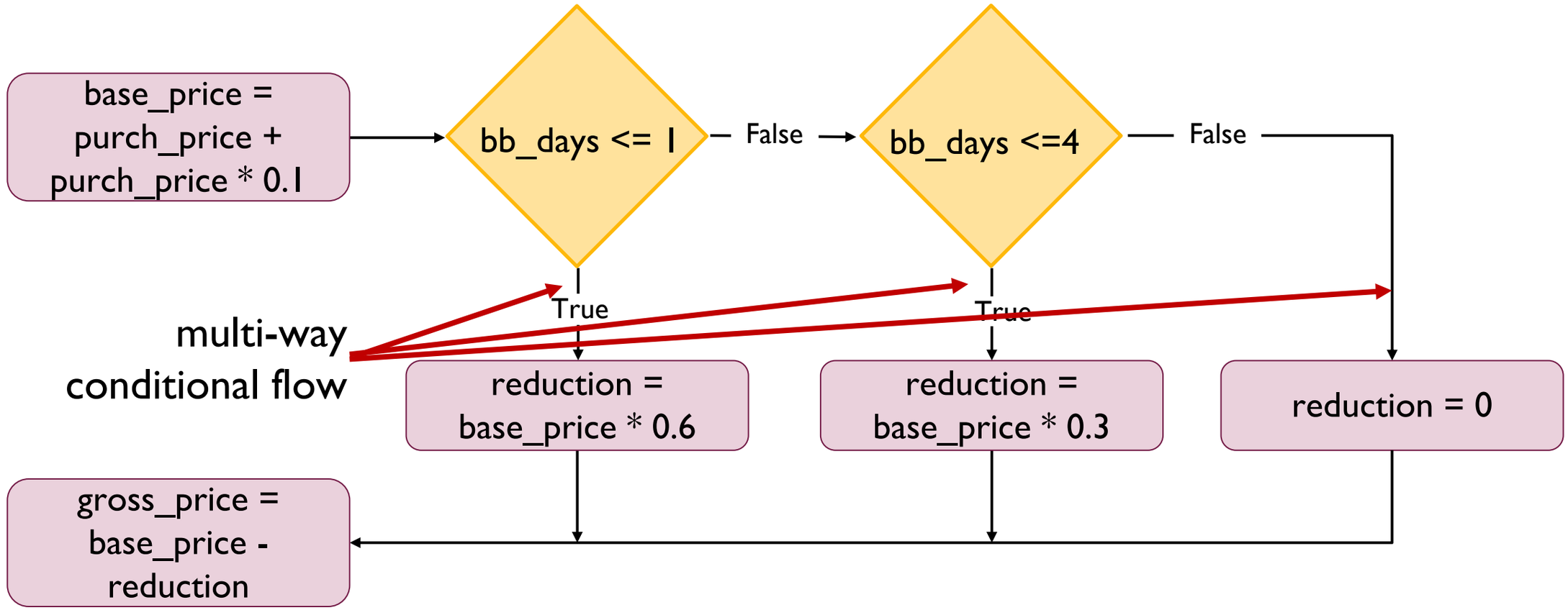
  30% reduction rate if within 3 days

  60% reduction rate if within 1 day
- sales prices has to incorporate goods and service tax (GST)

Can now use GST function in solution for overall problem



Now we need 3-way conditional



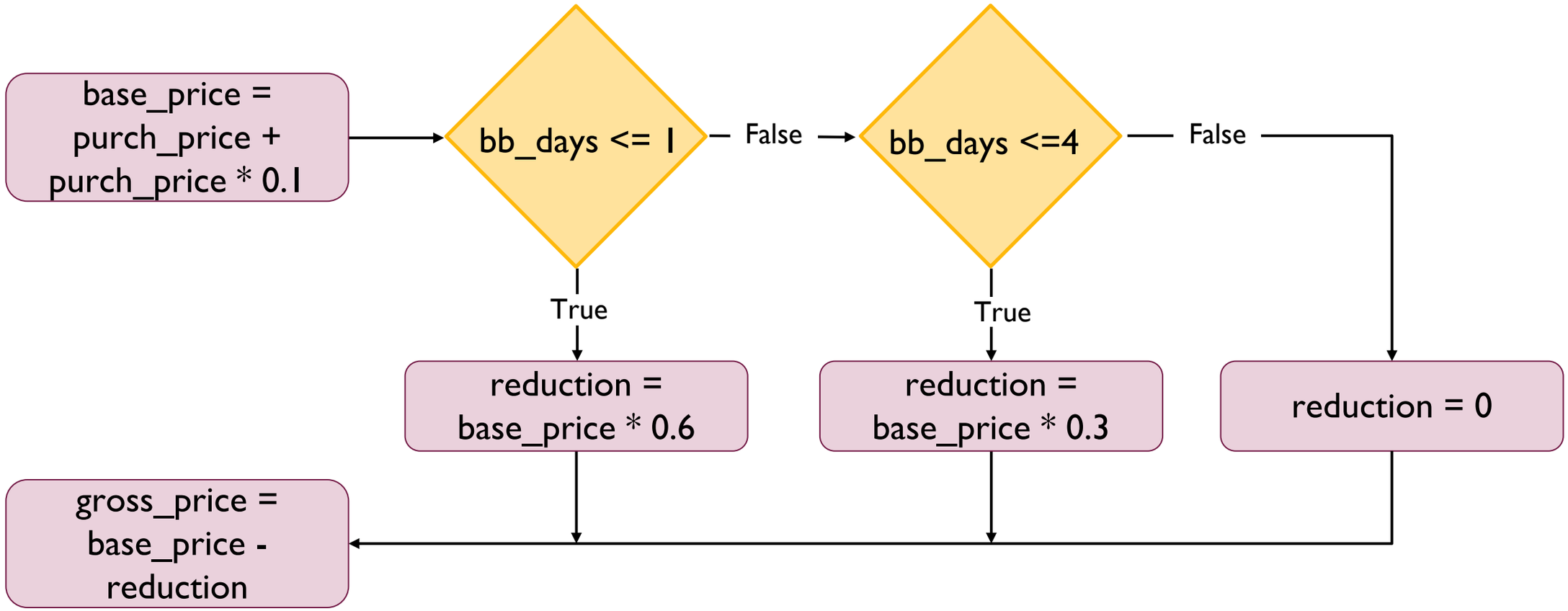

```
def customer_price(purch_price, product, best_before_days):
    base_price = purch_price + purch_price*0.1

        reduction = base_price*0.6

        reduction = base_price*0.3

        reduction = 0
    gross_price = base_price - reduction
    return round(price_after_gst(gross_price, product), 2)
```

Use **elif** for more alternatives




```
def customer_price(purch_price, product, best_before_days):
    base_price = purch_price + purch_price*0.1
    if best_before_days <= 1:
        reduction = base_price*0.6
    elif best_before_days <= 4:
        reduction = base_price*0.3
    else:
        reduction = 0
    gross_price = base_price - reduction
    return round(price_after_gst(gross_price, product), 2)
```

Recommended Reading

"*Introduction to Computing using Python: An Application Development Focus*", L. Perkovic

- **Sections 2.1 – 2.2 (types, expressions, vars)**
- **Sections 3.1 – 3.3 (conditionals, functions)**
- **Sections 5.2 – 5.6 (prep for tomorrow)**

FIT1045/53 Workbook

- **Chapter 1** (§1.2 Boolean Expressions)
- **Chapter 2** (§2.1 Conditionals)

On Wednesday…

- Loops and repetition!
- Loops and repetition!
- Loops and repetition!
- Loops and repetition!
- Loops and repetition!
- Loops and repetition!
- ….