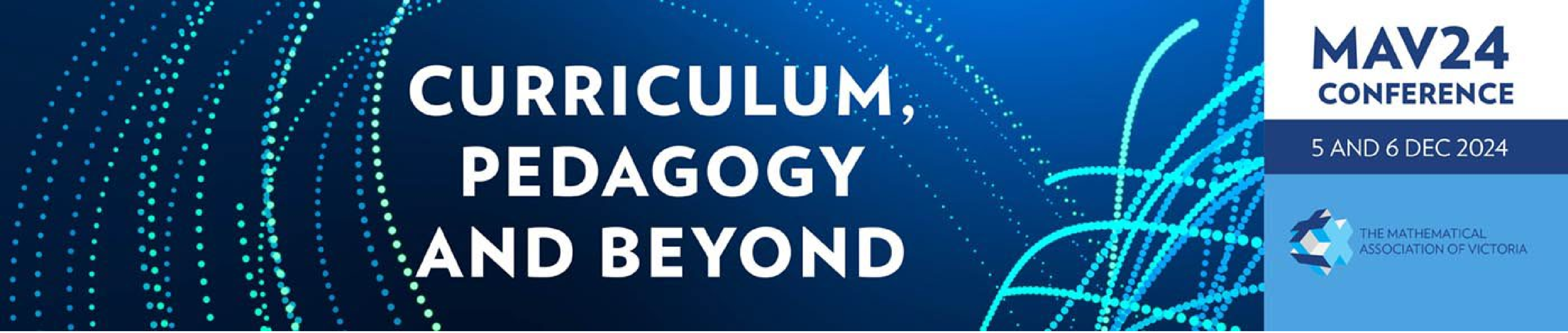


Implementing pseudocode and algorithms in Python on computer and CAS

Presented by Enzo Vozzo, 5 & 6 Dec 2024

Mathematics Teacher at Mentone Grammar

A25 IMPLEMENTING PSEUDOCODE AND ALGORITHMS IN PYTHON ON COMPUTER AND CAS

Subtheme: Technology

Enzo Vozzo, Mentone Grammar
(Year 7 to Year 12)

The introduction of Pseudocode in the new Mathematical Methods and Specialist Mathematics Study Design indicates that algorithms and coding are beginning to be seen as important. This presentation introduces the three key elements of algorithm design: sequencing, decision-making and repetition. These elements will be implemented using the popular open-source computer language Python on a computer and on the new TI CAS Nspire CX II calculator, which has Python built into it. Delegates will have the choice of coding a variety of simple algorithms to calculate the value of pi (using the bisection method), generate Pythagorean triples and primes, run simulations and define (create your own) mathematical functions such as factorials, sine and square roots. Python also handles complex numbers, with the ability to calculate Euler's identity in a single line of code! No experience of coding or Python is required but would be beneficial.

Key takeaways:

1. Introduction to pseudocode and algorithm design: sequencing, decision-making and repetition.

2. Introduction to the popular open-source computer language Python.

3. Choice of writing code to calculate the value of pi, generate Pythagorean triples, primes, run simulations, and define functions such as sine and square roots in terms of elementary arithmetic.

Remember: Delegates do need to have Python installed on their computer or it can be installed from the Python.org website or use a web-based version. Delegates should bring their laptop and/or TI CAS Nspire CX II which has Python built in. Python is not available on the Casio ClassPad FX-CP400.

Part 1 of 5: Pseudocode

Part 2 of 5: Python

Core Presentation

Options

Part 3 of 5: Implementing Pseudocode in Python

Part 4 of 5: Implementing Python on the TI CAS

Part 5 of 5: Free online resources for learning Python

Part 1 of 5: Pseudocode

Pseudocode in the new VCE Mathematical Methods and Specialist Mathematics Study Designs

Pseudocode

https://www.vcaa.vic.edu.au/curriculum/vce/vce-study-designs/Pages/PseudoCode.aspx

Mathematical Methods Study Design

https://www.vcaa.vic.edu.au/news-and-events/professional-learning/VCE/Pages/VCEMathematicalMethodsWebinars.aspx

Specialist Mathematics Study Design

https://www.vcaa.vic.edu.au/news-and-events/professional-learning/VCE/Pages/VCESpecialistMathematicsWebinars.aspx

Topic 7 – Content conducive for pseudocode

Mathematical Methods

- Bisection
- Newton's Method for polynomials and other functions
- Simple simulations in probability
- Numerical integration (e.g. trapezium method)

Specialist Mathematics

- Numerical integration (e.g. Reimann sums)
- Investigation of sequences
- Vector operations (e.g. cross product)
- Sample distributions for means

Pseudocode:

A plain language description of the steps in an algorithm.

Uses structural conventions of a programming language.

To be read by humans, not machines.

Pseudocode can be translated into real code.

Algorithm:

A set of instructions aimed at achieving a task.

Muḥammad ibn Mūsā al-Khwārizmī (c. 780 – c. 850)

Reserved or key words in Pseudocode

```
Define  Return

Input   Print

If … Then … Else … Else If … EndIf

For … From … To …    EndFor

While … Do …         EndWhile
```

The equals sign and the arrow symbol

`sum = sum + 1`

`sum ← sum + 1`

equivalent statements

The new sum equals the current sum + 1.

The current sum + 1 goes to new sum.

Both mean increment the variable sum by 1.

Pseudocode versus **Flow chart**

An *informal high-level description* of the operating principle of an algorithm.

Written in natural language and mathematical symbols.

A *diagrammatic representation* that illustrates a solution model to a given problem.

Written using various symbols.

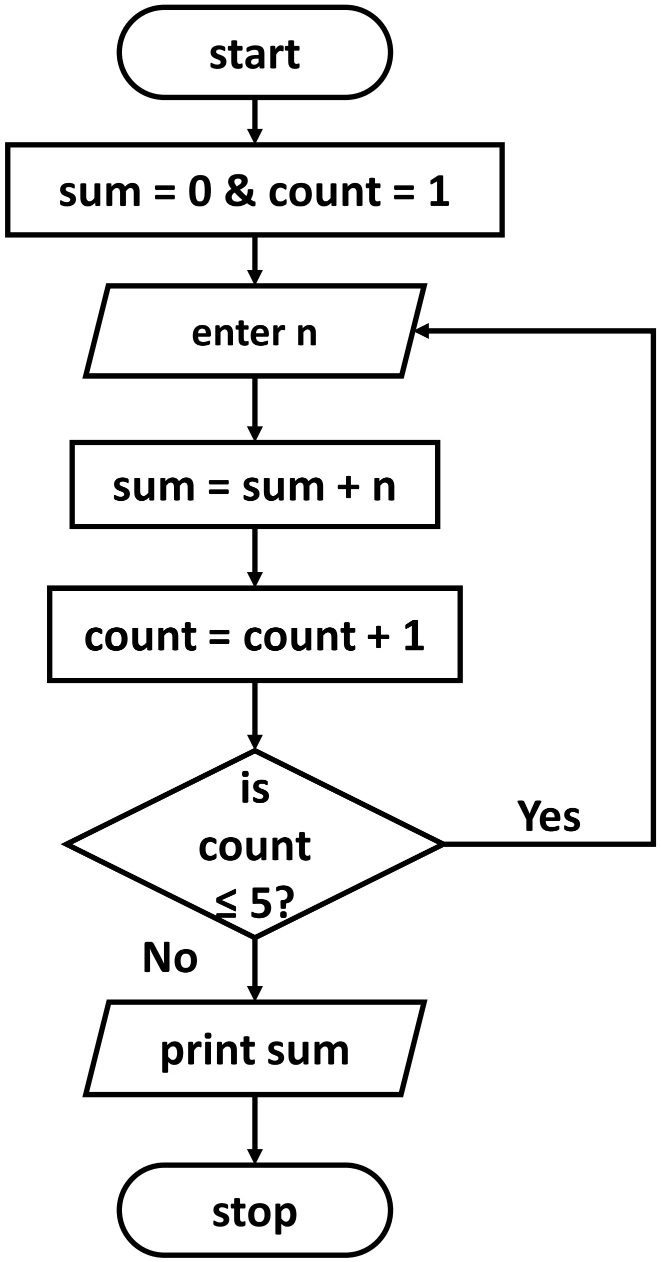
Example: Add five numbers:

Pseudocode

```
sum ← 0
count ← 1
While count ≤ 5 Do
    enter n
    sum ← sum + n
    count ← count + 1
Print sum
```

Indent by using Tab,
usually about 4 spaces.

Flow chart



The ***three*** key elements of algorithm design:

1. **Sequencing**
2. **Decision-making**
3. **Repetition (iteration)**

These three elements ***can be arranged in a variety of ways*** to achieve an outcome.

This is what allows code and computers to do a myriad of different things.

1. Sequencing

A series of statements.

```
Input a
Input b
sum ← a + b
difference ← a – b
product ← a * b
quotient ← a / b
Print sum, difference, product, quotient
```

2. Decision-making

Deciding on a course of action(s) depending on the state of a condition being true or false.

```
If a condition is TRUE Then
        …
        …
        …
Else
        …
        …
        …
EndIf
```

Any number of statements. (the statements between `If … Then` and `Else`)

The `Else` part is optional.

Any number of statements. (the statements between `Else` and `EndIf`)

Note the indentation makes the `if` structure easier to read.

3. Repetition (iteration)

Repeating the same statement(s), depending on the state of a condition.

```
For i From 1 To 10
      …
      …
      …
      …
EndFor
```

These statements are repeated 10 times.

A `for` loop has a ***<u>fixed</u>*** number of repetitions.

Note the indentation makes the `for` structure easier to read.

3. Repetition (iteration)

Repeating the same statement(s), depending on the state of a condition.

```
While a condition is TRUE Do
        …
        …
        …
        …
EndWhile
```

These statements are repeated while a condition is true.

A **`while`** loop has a ***<u>variable</u>*** number of repetitions.

Note the indentation makes the **`while`** structure easier to read.

Mathematical operation symbols

Add **+**

Subtract **–**

Multiplication *****

Division **/**

Exponentiation **^**

Equals **=** or **←**

Bracket (parentheses) **( )**

Greater than **>**

Greater than or equal to **≥**

Less than **<**

Less than or equal to **≤**

Not equal to **≠**

Logical operations: **`and`, `or`, `not`**

Defining functions

Functions are sections of code that are called upon the perform specific tasks numerous times.

If a program is going to do the same function multiple times, the function is defined once before it is called and then can be called when required.

Defining functions

Functions
must be defined
BEFORE
they are called.

Example:

```
define f(x)
     return 3*x+2

# calling code
...
...
a = f(5)
...
b = f(7)
...
```

Results:

```
a = f(5) = 3(5)+2 = 17
b = f(7) = 3(7)+2 = 23
```

Assessing pseudocode

Assessing the understanding of pseudocode

Students will ***NOT*** be required to write pseudocode from scratch.

Students **may** be required to *analyse* what a section of pseudocode (or a particular line of pseudocode) is doing.

Students **may** be required to fill in some missing statement(s) to *complete* a section of pseudocode.

Students **may** be required to *debug* a piece of pseudocode, (i.e. identify an error in a section of pseudocode & correct it.)

https://www.vcaa.vic.edu.au/Documents/exams/mathematics/mathmethods2-samp-w.pdf 2023

The algorithm below, described in pseudocode, estimates the value of a definite integral using the trapezium rule.

```
Inputs: f(x), the function to integrate
        a, the lower terminal of integration
        b, the upper terminal of integration
        n, the number of trapeziums to use

Define trapezium(f(x),a,b,n)
    h ← (b - a) ÷ n
    sum ← f(a) + f(b)
    x ← a + h
    i ← 1
    While i < n Do
        sum ← sum + 2 × f(x)
        x ← x + h
        i ← i + 1
    EndWhile
    area ← (h / 2) × sum
    Return area
```

Consider the algorithm with the following inputs.

trapezium($\log_e$(x),1,3,10)

The value of the variable **sum** after **one** iteration of the **While** loop would be closest to

A. 1.281
B. 1.289
C. 1.463 ← Answer is C.✓
D. 1.617
E. 2.136

Consider the algorithm below, which uses the bisection method to estimate the solution to an equation in the form $f(x) = 0$.

https://www.vcaa.vic.edu.au/Documents/exams/mathematics/mathmethods2-samp-w.pdf

```
Inputs: f(x), a function of x in radians
        a, the lower interval endpoint
        b, the upper interval endpoint
        max, the maximum number of iterations

Define bisection(f(x),a,b,max)
    If f(a) × f(b) > 0 Then
        Return "Invalid interval"
    i ← 0
    While i < max Do
        mid ← (a + b) ÷ 2
        if f(mid) = 0 Then
            Return mid
        Else If f(a) × f(mid) < 0 Then
            b ← mid
        Else
            a ← mid
        i ← i + 1
    EndWhile
```

2023

The algorithm is implemented as follows.

```
bisection(sin(x),3,5,2)
```

Which would be returned when the algorithm is implemented as given?

A. -0.351
B. -0.108
C. 3.25
D. 3.5 ← **Answer is D.✓**
E. 4

https://www.vcaa.vic.edu.au/Documents/exams/mathematics/mathmethods2-samp-w.pdf

One way of implementing Newton's method using pseudocode, with a tolerance level of 0.001, is shown below.
The pseudocode is incomplete, with two missing lines indicated by an empty box.

2023

```
Inputs: f(x), a function of x
        x0, an initial estimate
        for the x-intercept of f(x)

Define newton(f(x), x0)
  df f(x) ← the derivative of f(x)
  i ← 0
  prev_x ← x0
  While i < 1000 Do
    next_x ← prev_x − f(prev_x) ÷ df(prev_x)
  [                                         ]
    Else
        prev_x ← next_x
        i ← i + 1
    EndWhile
```

Which one of the following options would be most appropriate to fill the empty box?

A.
```
If next_x − prev_x < 0.001 Then
    Return prev_x
```

B.
```
If next_x − prev_x < 0.001 Then
    Return next_x
```

C.
```
If prev_x − next_x < 0.001 Then
    Return next_x
```

D.
```
If −0.001 < next_x − prev_x < 0.001 Then
    Return prev_x
```

E.
```
If −0.001 < next_x − prev_x < 0.001 Then
    Return next_x
```

Answer is E. ✓ →

The procedure below has been written in pseudocode. **2023** https://www.vcaa.vic.edu.au/Documents/exams/mathematics/specmath2-samp-w.pdf

```
declare integer n
declare integer f
declare integer t1
declare integer t2

set f to 0
set t1 to 2
set t2 to 3
set n to 3

repeat n times
    f = t1 + 2 × t2
    t2 = f
    print f
end loop
```

The output of the pseudocode is a list of numbers.
The final number in the list is

A. 3
B. 18
C. 38 ⟵ **Answer is C.** ✓
D. 72
E. 78

$when\ n = 1:$

$f = 2 + 2 \times 3 = 8$

$t2 = f = 8$

$when\ n = 2:$

$f = 2 + 2 \times 8 = 18$

$t2 = f = 18$

$when\ n = 3:$

$f = 2 + 2 \times 18 = 38$

$t2 = f = 38$

$print\ f = 38$

2024

Question 17

Consider the algorithm below, which prints the roots of the cubic polynomial $f(x) = x^3 - 2x^2 - 9x + 18$.

```
define f(x)
    return (x³ - 2x² - 9x + 18)
c ← f(0)
if c < 0 then
    c ← (-c)
end if
while c > 0
    if f(c) = 0 then
        print c
    end if
    if f(-c) = 0 then
        print -c
    end if
    c ← c - 1
end while
```

In order, the algorithm prints the values

A. $-3, 3, 2$

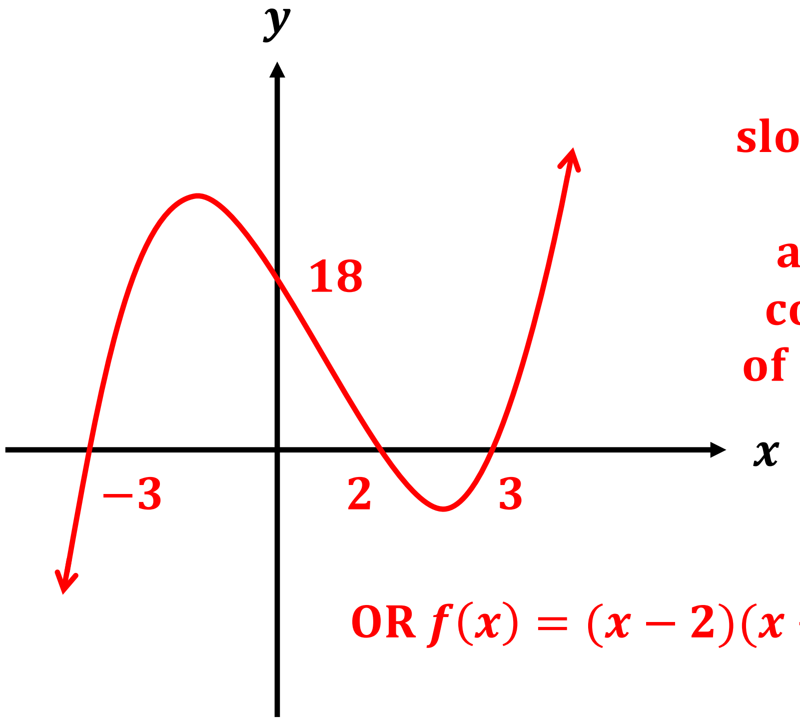
B. $-3, 2, 3$

C. $3, 2, -3$

D. $3, -3, 2$

$f(x) = x^3 - 2x^2 - 9x + 18$

By inspection, $f(x)$ is a positive cubic and has the shape →



slope @ $x = 0$ is -9, as per the coefficient of the x term.

OR $f(x) = (x-2)(x+3)(x-3)$

And the roots in ascending order are: $-3, 2, 3$

But this is not the order in which the algorithm prints them.

The next slide explains the algorithm.

Question 17

Consider the algorithm below, which prints the roots of the cubic polynomial $f(x) = x^3 - 2x^2 - 9x + 18$.

```
define f(x)
    return (x^3 - 2x^2 - 9x + 18)
c ← f(0)
if c < 0 then
    c ← (-c)
end if
while c > 0
    if f(c) = 0 then
        print c
    end if
    if f(-c) = 0 then
        print -c
    end if
    c ← c - 1
end while
```

In order, the algorithm prints the values

A. $-3, 3, 2$

B. $-3, 2, 3$

C. $3, 2, -3$

D. $3, -3, 2$

The algorithm with its comments which explains what it's doing.

2024

Define f(x) = x^3 - 2x^2 - 9x + 18

Set c to the value of f(0) = 18

If c is negative, make it positive. This positive value will be used as the starting value for finding the integer roots to f(x).

Beginning with c = 18, the algorithm tests to see if f(18)=0?, f(-18)=0?, f(17)=0?, f(-17)=0?, f(16)=0? f(-16)=0?, ... f(2)=0?, f(-2)=0?, f(1)=0?, f(-1)=0?. and prints the value of c or -c only when f(c)=0 or f(-c)=0.

It does not test f(0)=0 because f(0)=18. This is the y-intercept.

It finds these three roots in the order: f(3)=0, f(-3)=0, f(2)=0.

Hence, the algorithm prints the roots in the following order: 3, -3, 2.

These are the integer roots of f(x).

The answer is option D: 3, -3, 2

Part 2 of 5: Python

The computer language Python

Creator of Python:
Guido van Rossum
(1956 –)

Very popular open-source programming language.

Easy to learn.

Python code reads similar to pseudocode.

Python has excellent resources at: https://www.python.org/

Reserved or key words in Python

`def` `return`	Defining functions	**Number types:**
`input` `print`	Input and output	`int` (integers)
`if` `else` `elif`	Decision making	
`for ... in range` `while`	Iterations	`float` (decimals)

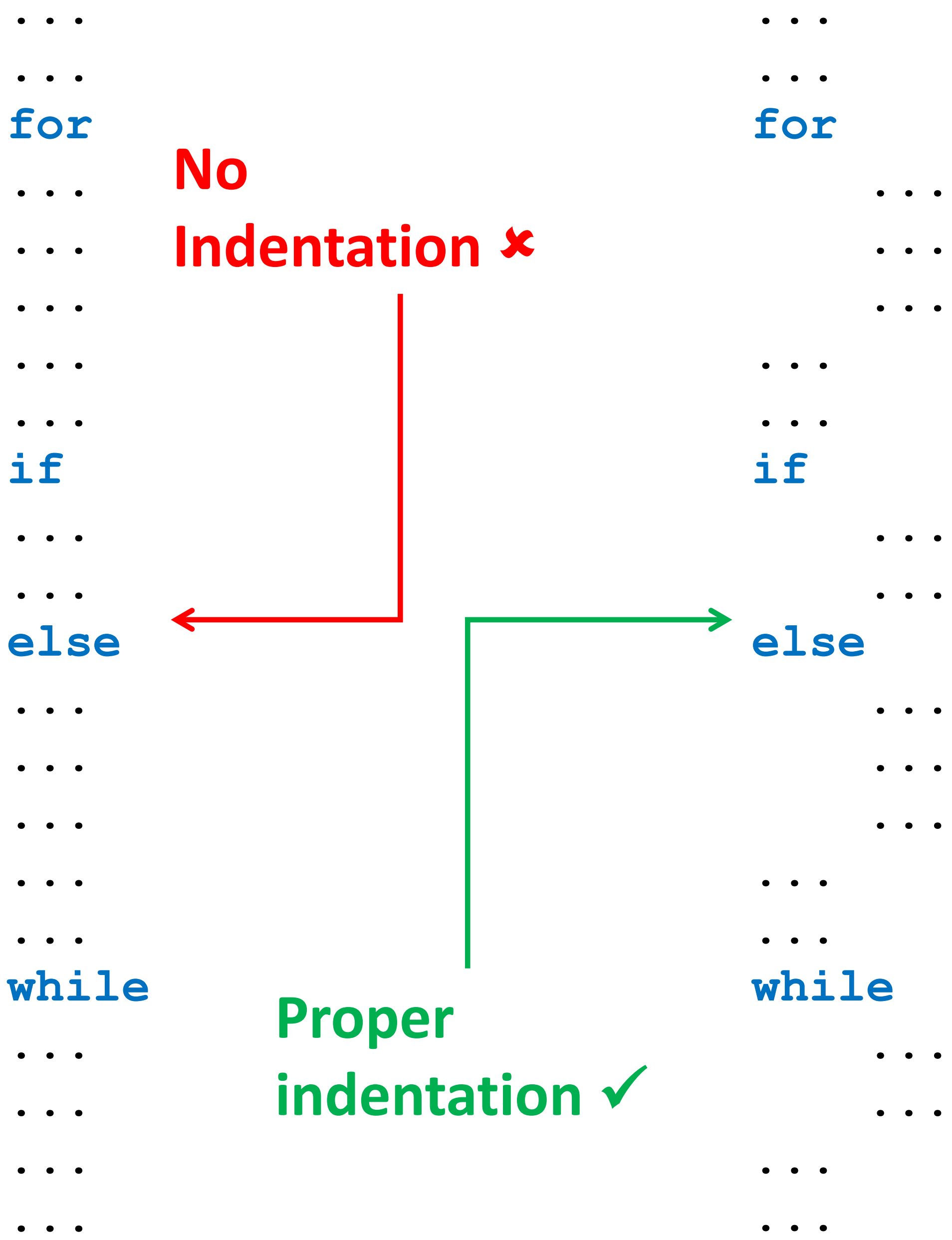
Indentation

Indentation makes pseudocode and real code more readable.

Python is very strict on correct indentation.

If you make an indentation error, Python will notify you of it and locate it for you to correct.

Errors MUST be corrected before a program will run.



The *three* key elements of algorithm design in Python:

1. **Sequencing**
2. **Decision-making**
3. **Repetition (iteration)**

The *three* key elements of algorithm design in Python:

1. **Sequencing**
2. Decision-making
3. Repetition (iteration)

```
pi = 3.14

x = a + b

y = 3 * x - 4

z = math.sqrt(3**2 + 4**2)
```

The ***three*** key elements of algorithm design in Python:

1. Sequencing
2. **Decision-making**
3. Repetition (iteration)

```
if x == 0:
    ...
    ...
    ...
else:
    ...
    ...
    ...
```

In an `if` statement, ***two*** equal signs are required as this is a comparison, not an assignment.

The *three* key elements of algorithm design in Python:

1. Sequencing
2. **Decision-making**
3. Repetition (iteration)

```
if x == 0:        if x != 1:        if x > 2:        if x <= 3:
    ...               ...               ...              ...
    ...               ...               ...              ...
    ...               ...               ...              ...
    ...               ...               ...              ...
```

The ***three*** key elements of algorithm design in Python:

1. Sequencing
2. Decision-making
3. **Repetition (iteration)**

```
for n in range(1, 10+1, 1): # repeat 10 times
    ...
    ...
    ...
    ...
```

Step size

Final number

Starting number

The *three* key elements of algorithm design in Python:

1. Sequencing
2. Decision-making
3. **Repetition (iteration)**

```
while x < 100: # repeat while x < 100
    ...
    ...        x must reach the value of 100
    ...        otherwise, the loop will be infinite
    ...
```

Inputting in Python

User → Python

```python
a = float(input("a = "))
# get a floating-point number from the user
# and assign it to the variable a.


b = int(input("b = "))
# get an integer value from the user
# and assign it to the variable b.
```

Note that the number of open brackets
MUST equal the number of closed brackets.

Outputting in Python

User ← Python

```
print(a, b, a + b)
# print the variables a, b and the sum a + b

print("a = ", a)
# print the string of text "a = " and
# the actual value of a next to this

print()    # print a blank line
```

Comments in Python

Comments make real code easier to understand.

Comments are preceded by the hash symbol (#)

Example:

```
x = int(input("x ="))        # get the first number
y = int(input("y ="))        # get the second number
z = x + y                    # add the numbers
print("z = ", z)             # display the result
```

Python code

Python comments

Commenting out sections of Python code

Example:

	`Statement 1`	Executed	
	`Statement 2`	Executed	
Start of comments three apostrophes	`'''`	“”	
	`Statement 3`	Not executed	Skipped
	`Statement 4`	Not executed	
	`Statement 5`	Not executed	
End of comments three apostrophes	`'''`	“”	
	`Statement 6`	Executed	
	`Statement 7`	Executed	
	`Statement 8`	Executed	
	`Statement 9`	Executed	

Python mathematical operation symbols

Add **+**

Subtract **–**

Multiplication *****

Division **/** or **%**

Exponentiation ****** **(not ^)**

Equals **=**

Brackets (parentheses) **( )**

Greater than **>**

Greater than or equal to **>=**

Less than **<**

Less than or equal to **<=**

Not equal to **!=**

Logical operations: **and**, **or**, **not**

Python mathematical predefined functions

```python
import math # required for mathematical functions
```

absolute value: **`abs(x)`** $|x|$

square root: **`math.sqrt(x)`** $\sqrt{x}$

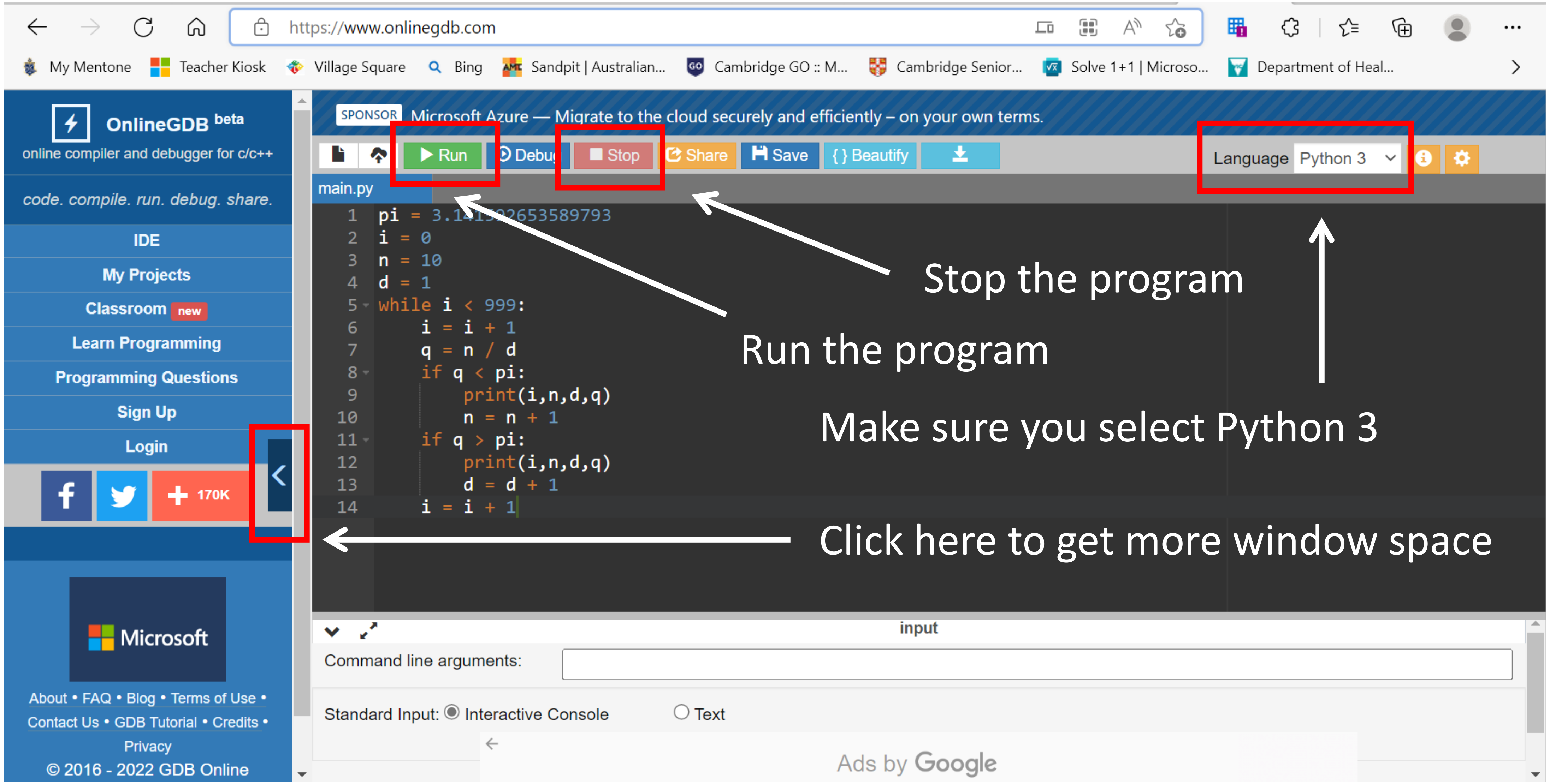
factorial : **`math.factorial(n)`** $n!$

Python mathematical predefined functions

```python
import math # required for mathematical functions
```

sine:	`math.sin(x)`	$\sin(x)$
cosine:	`math.cos(x)`	$\cos(x)$
tangent:	`math.tan(x)`	$\tan(x)$
exponential:	`math.exp(x)`	e^x
natural logarithm:	`math.ln(x)`	$\ln(x)$
logarithm base 10:	`math.log10(x)`	$\log_{10}(x)$

https://www.onlinegdb.com/



Euclidian algorithm to find the GCD of a pair of numbers

GCD Pseudocode	GCD Python code
Define gcd(a, b)	def gcd(a, b):
While a ≠ b Do	while a != b:
If a > b Then	if a > b:
a ← a – b	a = a - b
Else	else:
b ← b – a	b = b - a
Return a	return a

```
calling code
a = 48
b = 64
Print(a, b, gcd(a, b))
```

```python
# driver code
a = 48
b = 64
print(a, b, gcd(a, b))
```

Part 3 of 5:

Implementing pseudocode in Python on a computer

Delegates can select which algorithms they want to code.

List of Python programs

SIMPLE ALGORITHMS:

- Swap algorithm
- Euclid's Greatest common divisor algorithm
- Generating random numbers for simulations

Pi algorithms:

- **Madhava-Gregory-Leibniz formula**
- **Newton's method**
- **Bisection method**
- **Monte Carlo simulation method**

e algorithm:

- **e = 1/0! + 1/1! + 1/2! + 1/3! + 1/4! + ...**

MISCELLANEOUS ALGORITHMS

- Generating Pythagorean triples
- List primes
- Factorial function
- Square root function

Sine function and cosine function:
These are used to generate a sin, cos and tan table of values from 0, 15, 30, 45, ..., 360

Integration:
- **Trapezium rule**
- **Riemann sum**

Complex numbers calculations

```python
# swap algorithm

x = float(input("x = ")) # get the 1st number
y = float(input("y = ")) # get the 2nd number
print(x, y)  # print the numbers before the swap
x = x + y
y = x - y
x = x - y
print(x, y) # print the numbers after the swap
```

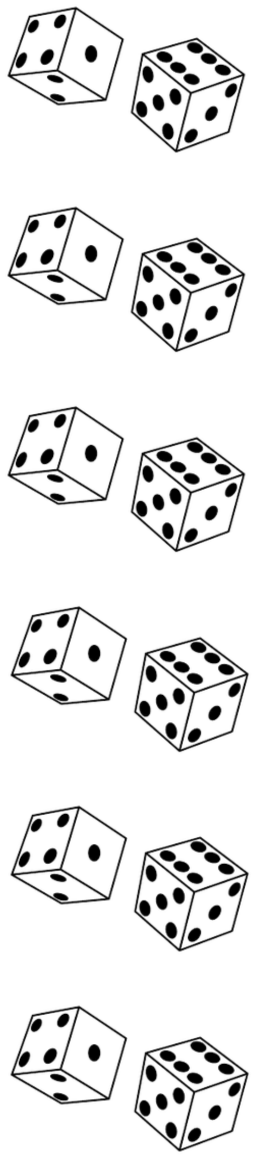
The actual algorithm.

Note that this algorithm does NOT require a temporary variable.

```python
# Euclid's Greatest common divisor algorithm

def gcd(a, b):
    while a != b:        # != means ≠
        if a > b:
            a = a - b
        else:
            b = b - a
    return a

# driver code
a = 48
b = 64
print(a, b, gcd(a, b))
```

Euclid
(fl. 300 BC)

```python
# generating random numbers, integers and decimals

import random

# random.seed(1) # generate the same random numbers

for i in range(1, 5+1, 1):    # loop 5 times
    x = random.randint(0, 9) # 0 ≤ x ≤ 9 integer
    y = random.random()       # 0 ≤ y ≤ 1 decimal
    print(i, x, y)
```



Random Results:

```
1 6 0.06915053997455245
2 1 0.3046571134385604
3 4 0.7340190767728589
4 9 0.1538597662419427
5 1 0.8113823902195922
```

These random numbers can be used for simulation purposes. An example of this will be the *Monte Carlo simulation method* for calculating pi.

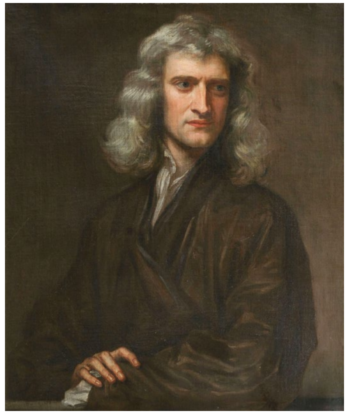
```
# Pi Madhava Gregory Leibniz formula

# pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - ...

pi = 0
den = 1
while 1: # infinite loop
    pi = pi + 4/den
    print(den, pi)
    den = den + 2
    pi = pi - 4/den
    print(den, pi)
    den = den + 2
```

Gottfried Wilhelm
(von) Leibniz
(1646 – 1716)

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \cdots$$

Converges to pi very slowly.

```python
# Newton's method for finding solutions to equations

import math

def f(x): # the function
    return math.sin(x) # sin(pi) = 0

def df(x): # the derivative of the function
    return math.cos(x)  # to find pi

i = 1      # iteration number
x = 3      # initial estimate
error = 1 # dummy error value
while error > 10e-9:
    xnext = x - f(x)/df(x)
    error = abs(xnext - x)
    x = xnext
    print(i, x)
    i = i + 1
```

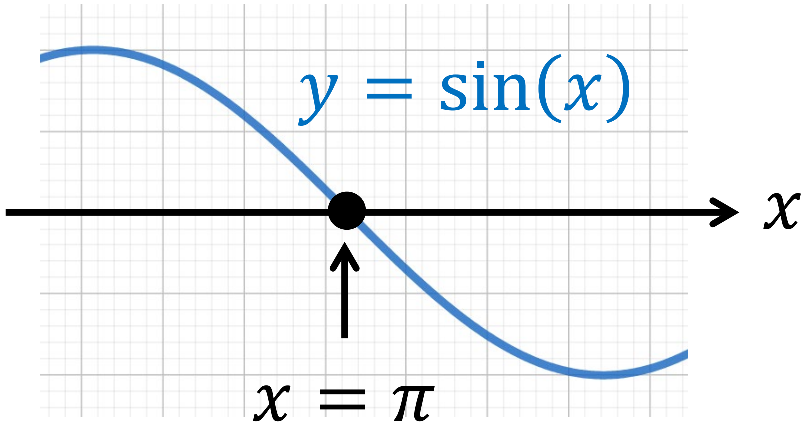


Isaac Newton
(1642 – 1728)

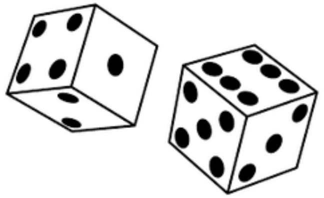
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

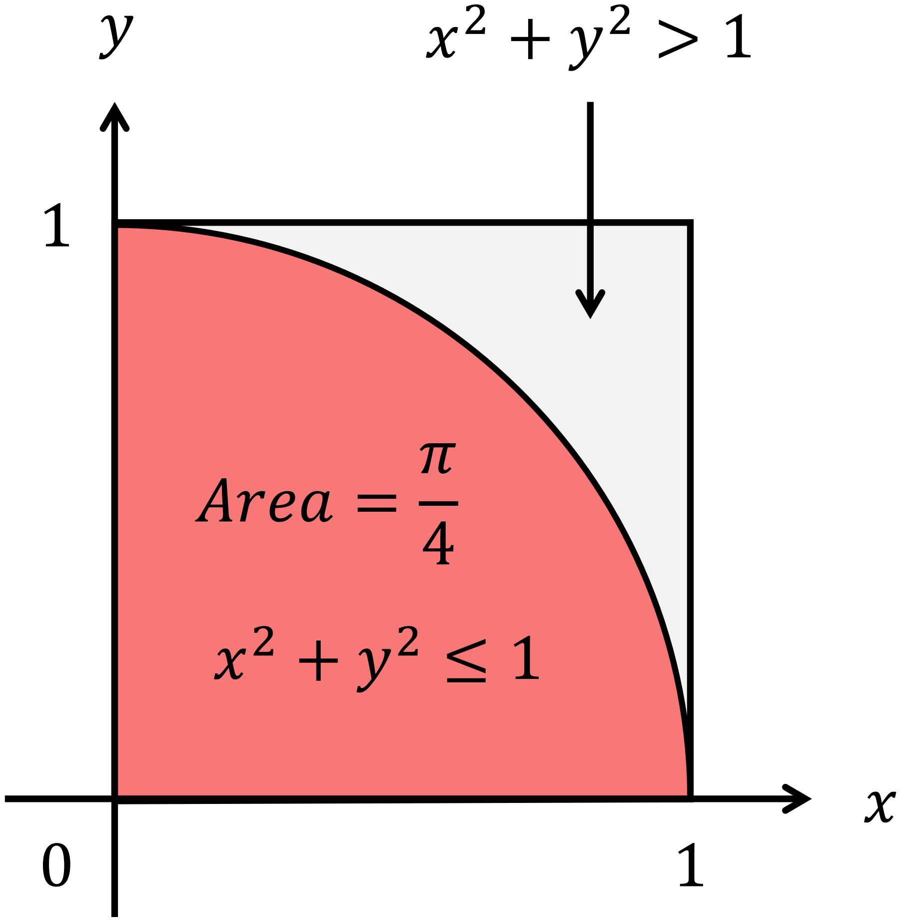
Converges to pi very quickly.

```python
# Bisection method to calculate pi
# The function sin(x) = 0 when x = pi

import math
a = 3; b = 4; i = 1
while i <= 60:
    c = (a + b) / 2
    if math.sin(a) * math.sin(c) < 0:
        b = c
    else:
        a = c
    print(i, "   ", c)
    i = i + 1
```



```python
# Pi using the Monte Carlo simulation method

import math
import random

random.seed()

count = 0
iteration = 0

while iteration < 1000000:
    x = random.random() # 0 <= x <= 1
    y = random.random() # 0 <= y <= 1
    sum = x * x + y * y
    if sum <= 1:
        count = count + 1
    iteration = iteration + 1
    pi = count / iteration * 4
    print(iteration, pi)
```





```python
# e Euler's number 2.7182818284590...

# e = 1/0! + 1/1! + 1/2! + 1/3! + 1/4! + ...

import math
e = 1
den = 1
while den <= 20:
    e = e + 1/math.factorial(den)
    print(den, e)
    den = den + 1
```

Converges to e very quickly.

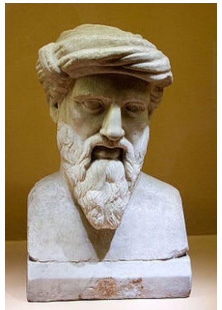
```python
# Euclid's algorithm to generate Pythagorean triples

import math
print("Generating Pythagorean triples:")
print("")
print("i    m n    a b c")
print("")
i = 0
maximum = 10
for m in range(2, maximum, 1):
    for n in range(1, m, 1):
        i = i + 1
        a = m**2 - n**2
        b = 2 * m * n
        c = m**2 + n**2
        print(i, "  ", m, n, "  ", a, b, c,)
```

$m \,\&\, n \in \mathbb{N} \;\; with \;\; m > n > 0$

$a = m^2 - n^2$

$b = 2mn$

$c = m^2 + n^2$



Pythagoras
(c. 570 – c. 495 BC)

Euclid
(fl. 300 BC)

```python
# List primes

# isPrime function
import math

def isPrime(n):
    if (n <= 1):
        return False
    for i in range(2, int(math.sqrt(n))+1):
        if (n % i == 0):
            return False
    return True

# List the primes up to 100
for i in range(1, 100+1, 1):
    if isPrime(i):
        print(i)
```

2, 3, 5, 7, 11, 13, 17, 19, ...

Euclid
(fl. 300 BC)

```python
# my factorial function

def factorial(n): # n >= 0
    if n == 0:
        return 1
    else:
        f = 1
        for i in range(1, n+1, 1):
            f = f * i
        return f

# driver code
nfPrev = 1
for a in range(0, 70+1, 1):
    nf = factorial(a)
    print(a, nf, nf/nfPrev)
    nfPrev = nf
```

$$n! = n(n-1)(n-2)(n-3) \times \cdots \times 3 \times 2 \times 1$$

```python
# my square root function

def sqrt(a):
    if a == 0:
        return 0
    else:
        error = 1
        x = a/2
        while abs(error) > 1e-12:
            xNext = (x + a/x)/2
            error = abs(xNext - x)
            x = xNext
        return x

# driver code
for a in range(1, 100+1, 1):
    print(a, sqrt(a))
```

$$sqrt(a) = \sqrt{a}$$

$$If \;\; x = \sqrt{a} \;\; then \;\; x_0 = \frac{a}{2}$$

$$and \;\; x_{n+1} = \frac{x_n + a/x_n}{2}$$

```python
def factorial(n): # sin and cos function require n!
    if n == 0:
        return 1
    else:
        f = 1
        for i in range(1, n+1, 1):
            f = f * i
        return f
```

$$n! = n(n-1)(n-2)(n-3) \times \cdots \times 3 \times 2 \times 1$$

```python
def sin(x):  # the sine function requires n!
    sinx = (x**1)/factorial(1) - (x**3)/factorial(3)
    for n in range(5, 99, 4):
        sinx = sinx + x**(n)/factorial(n) - x**(n+2)/factorial(n+2)
    return sinx
```

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

```python
def cos(x):  # the cosine function requires n!
    cosx = (x**0)/factorial(0) - (x**2)/factorial(2)
    for n in range(4, 100, 4):
        cosx = cosx + x**(n)/factorial(n) - x**(n+2)/factorial(n+2)
    return cosx
```

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Continued on the next slide …

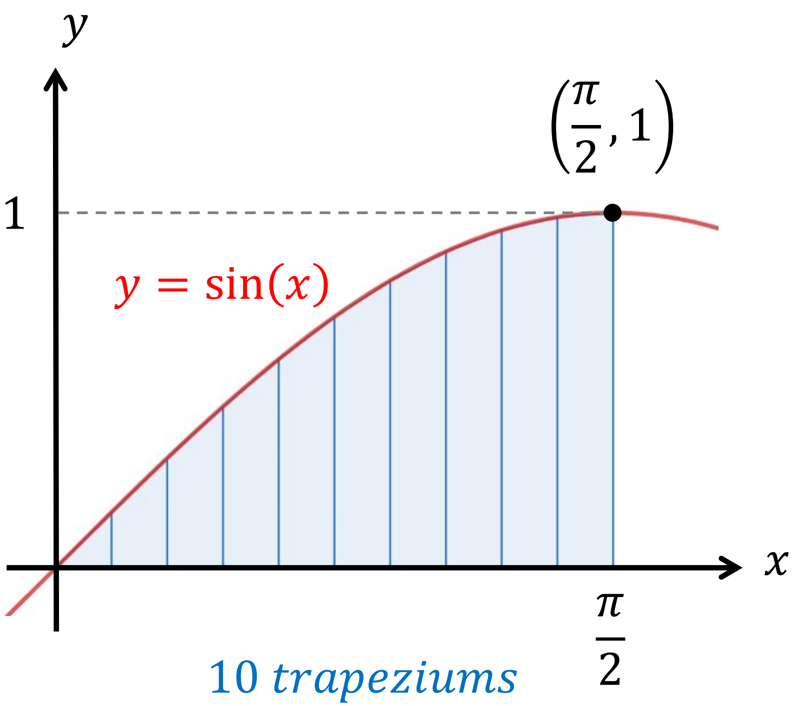
Continued from the previous slide …

```
# driver code
print("deg rad sin(deg) cos(deg) tan(deg)")
for d in range(0, 360+1, 15):
    r = d/180 * 3.141592653589793
    print(d, r, sin(r), cos(r), sin(r)/cos(r))
    # list sin cos tan in 15 degree increments
```

```
Results:

deg rad sin(deg) cos(deg) tan(deg)
0 0.0 0.0 1.0 0.0
15 0.2617993877991494 0.25881904510252074 0.9659258262890684 0.2679491924311226
30 0.5235987755982988 0.49999999999999994 0.8660254037844386 0.5773502691896257
45 0.7853981633974483 0.7071067811865475 0.7071067811865475 1.0
60 1.0471975511965976 0.8660254037844385 0.5000000000000001 1.7320508075688765
75 1.3089969389957472 0.9659258262890681 0.2588190451025207 3.7320508075688776
90 1.5707963267948966 1.0000000000000002 4.2539467343847745e-17 2.3507581604559628e+16
…
```

Pseudocode for integration using the **trapezium rule**

```
define f(x):
      return sin(x)

a ← 0
b ← 5
n ← 10
h ← (b - a)/n
left ← a
right ← a + h
sum ← 0
for i from 1 to n
      strip ← 0.5 * (f(left) + f(right)) * h
      sum ← sum + strip
      left ← left + h
      right ← right + h
end for

print sum
```



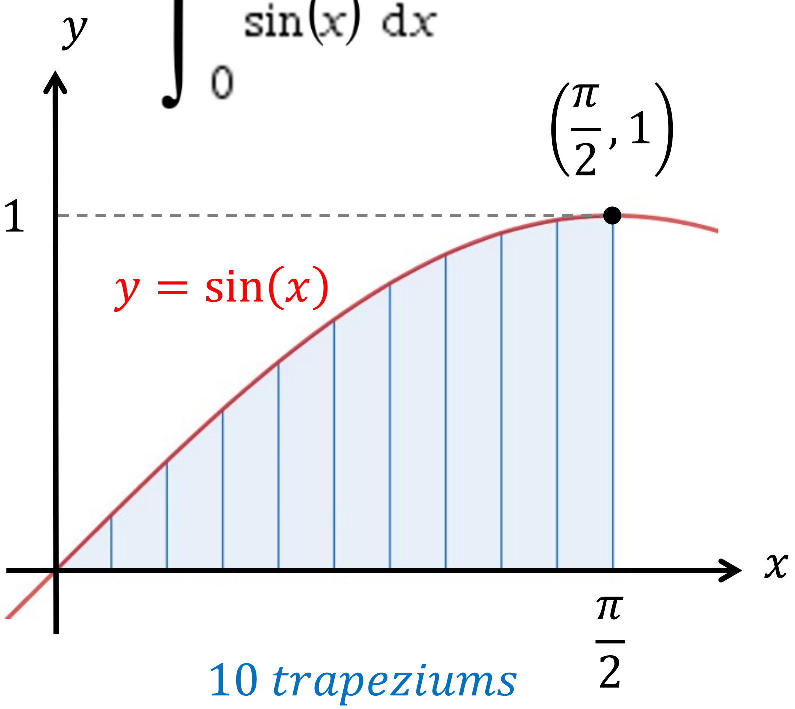
10 trapeziums

$Area = 0.9979429863544$

Python code for integration using the **trapezium rule**

```
# Integration: Trapezium rule

import math

def f(x):    # define the function
    return math.sin(x)

#driver code
a = 0              # lower limit of integration
b = math.pi/2      # upper limit of integration
n = 10             # number of trapezium strips
h = (b - a) / n    # height of each trapezium strip
left = a
right = a + h
sum = 0
for i in range (1, n+1, 1):
    strip = 1/2 * (f(left) + f(right)) * h
    sum = sum +  strip
    left = left + h
    right = right + h
print(sum)
```

$$\int_0^{\frac{\pi}{2}} \sin(x)\,dx \qquad 1.$$



10 trapeziums

$Area = 0.9979429863544$

```python
# Integration: Riemann Sum

import math

def f(x):                    # define the function
    return math.sin(x)   # f(x) = sin(x)

# driver code

a = 0                    # lower limit of integration
b = math.pi/2            # upper limit of integration
n = 100                  # number of trapezium strips

dx = (b - a) / n         # dx width of each rectangle
sum = 0
x = a
while x <= b:
    stripArea = f(x) * dx
    sum = sum + stripArea
    x = x + dx
print(sum)
```

Bernhard Riemann
(1826 – 1866)

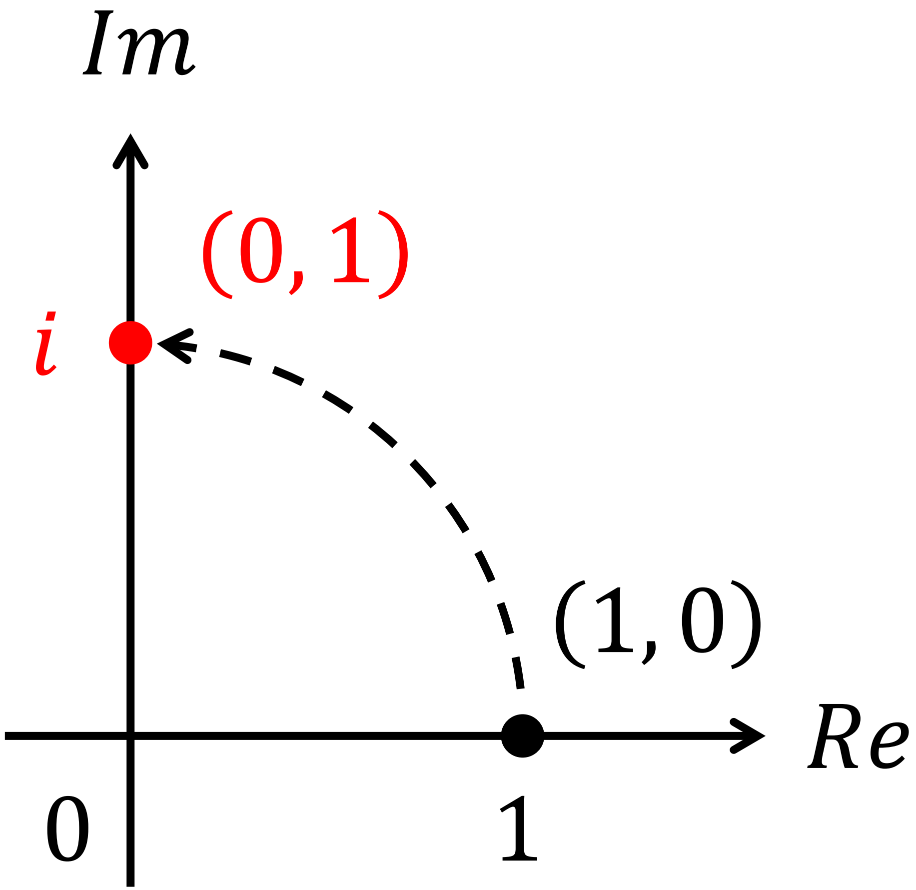
Python code for integration using a **Riemann sum**

$$\int_0^{\frac{\pi}{2}} \sin(x)\, dx \qquad 1.$$

Complex numbers

$\sqrt{-1} = i$

$(0, 1) = i$



Complex numbers

```
import cmath  # required for complex number calculations

e = cmath.e                       # define e

pi = cmath.pi                     # define pi

i = 1j                            # define i

i = complex(0, 1)                 # define i

i = cmath.sqrt(-1)                # define i

i = cmath.e**(1j*cmath.pi/2) # define i
```

Alternative definitions of i

Complex numbers

$\sqrt{-1} = i$

```
print(cmath.sqrt(-1))
```

$i^2 = -1$

```
print(i**2)
```

$e^{i\pi} = -1$

```
print(e**(i*pi))
```

$e^{i\pi} + 1 = 0$

```
print(e**(i*pi)+1)
```

← Euler's identity

$\left(-\frac{1}{2} + \frac{\sqrt{3}}{2} i\right)^3 = 1$

```
print((-1/2+(cmath.sqrt(3)/2)*i)**3)
```

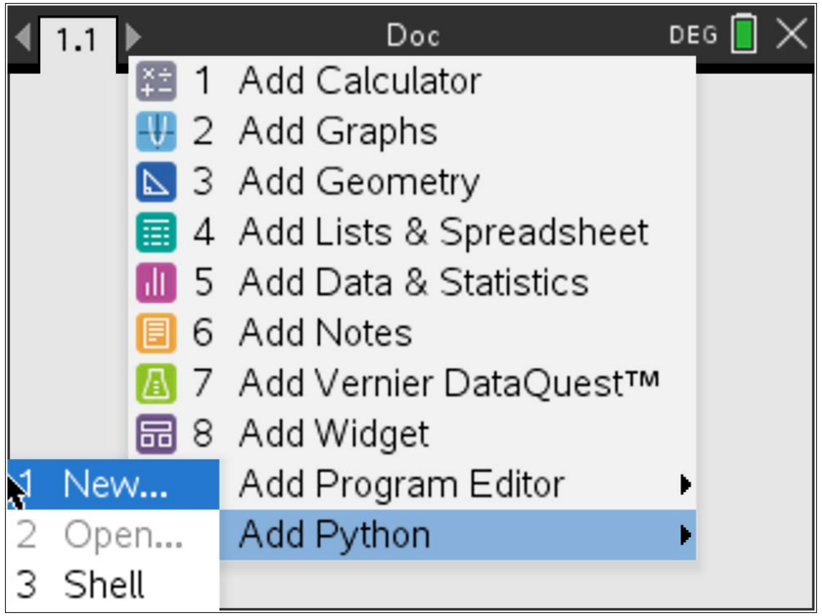
Part 4 of 5:

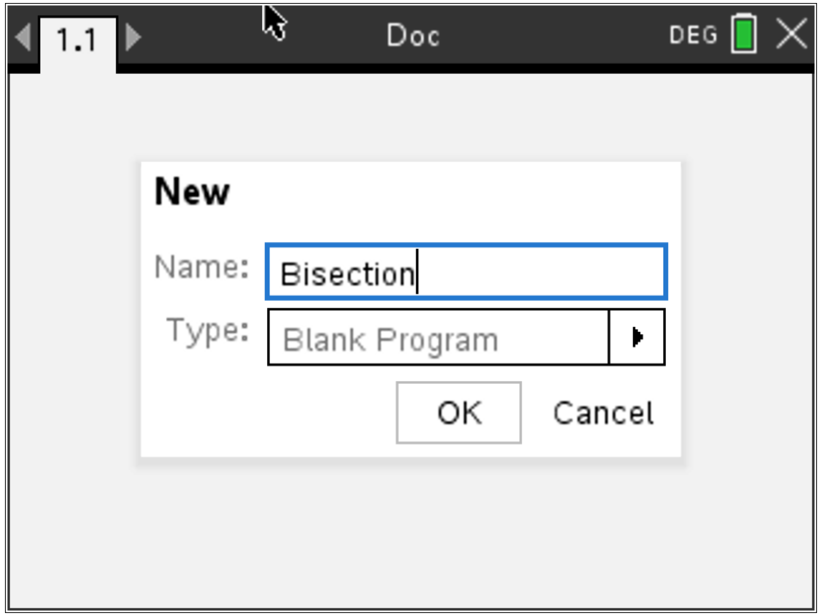
Implementing Python on the TI CAS

Python ***IS*** available on the new TI-*nspire* CX II CAS Calculator

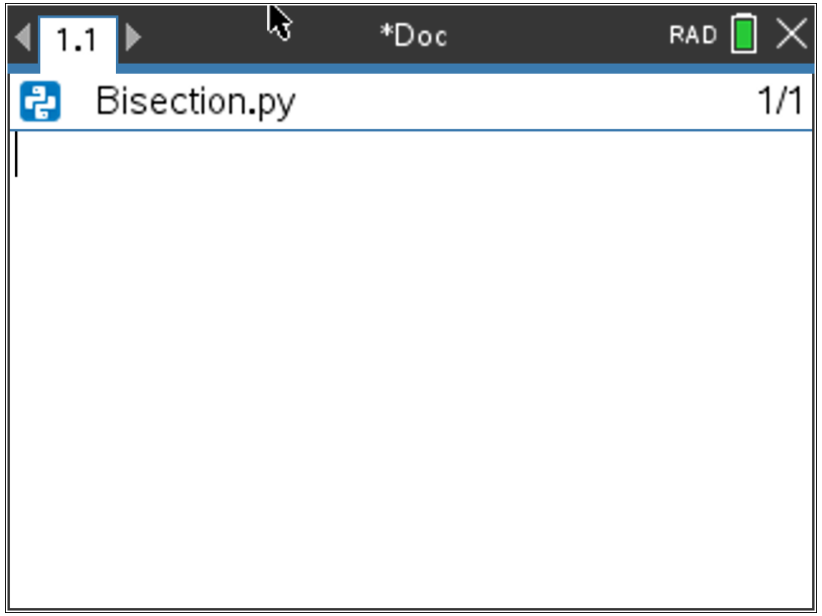
(Python is ***NOT*** available on the fx CP400 Casio ClassPad)

Bisection method to find the value of pi



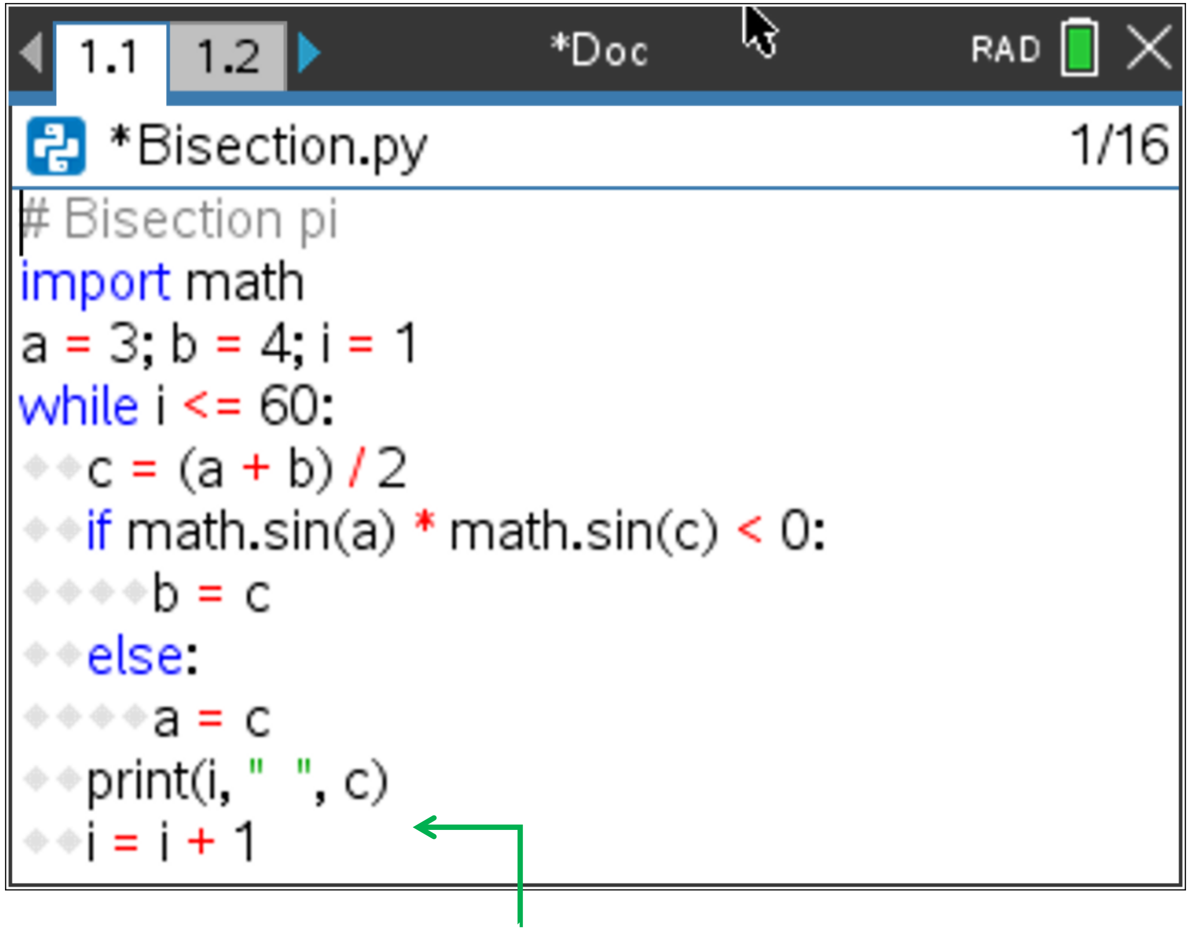


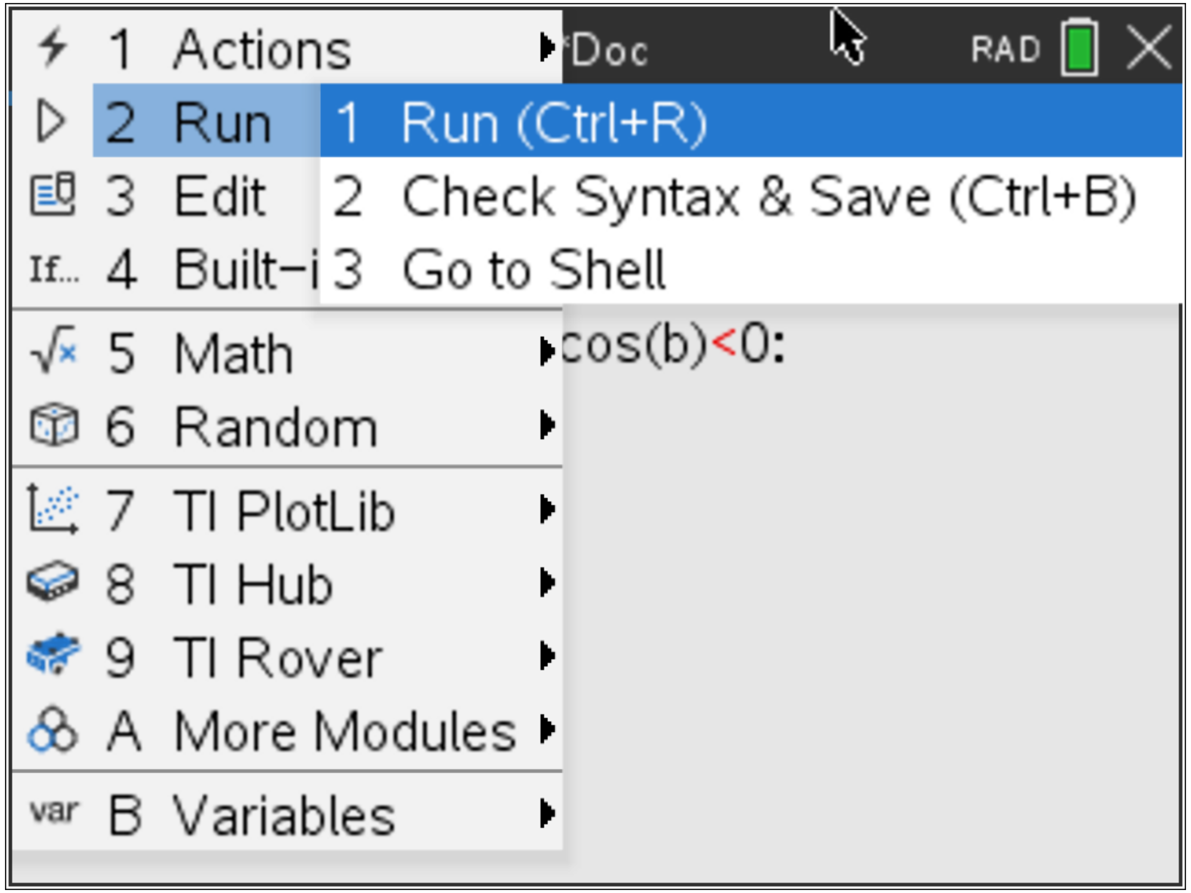
Start typing Python code here. →



Bisection method to find the value of pi

To run the code press menu 2 1 (Ctrl+R)






```
for d in range(1,10000,1):     # Optional delay to slow down the program.
    d=d+1-1*1/1
```

Bisection method to find the value of pi

Python Shell 1/63

```
>>>#Running Bisection.py
>>>from Bisection import *
1   3.5
2   3.25
3   3.125
4   3.1875
5   3.15625
6   3.140625
7   3.1484375
8   3.14453125
9   3.142578125
```

Python Shell 22/63

```
10   3.1416015625
11   3.14111328125
12   3.141357421875
13   3.1414794921875
14   3.14154052734375
15   3.141571044921875
16   3.141586303710938
17   3.141593933105469
18   3.141590118408203
19   3.141592025756836
20   3.141592979431152
```

Python Shell 33/63

```
21   3.141592502593994
22   3.141592741012573
23   3.141592621803284
24   3.141592681407928
25   3.141592651605606
26   3.141592666506767
27   3.141592659056187
28   3.141592655330896
29   3.141592653468251
30   3.141592654399574
31   3.141592653933913
```

Python Shell 44/63

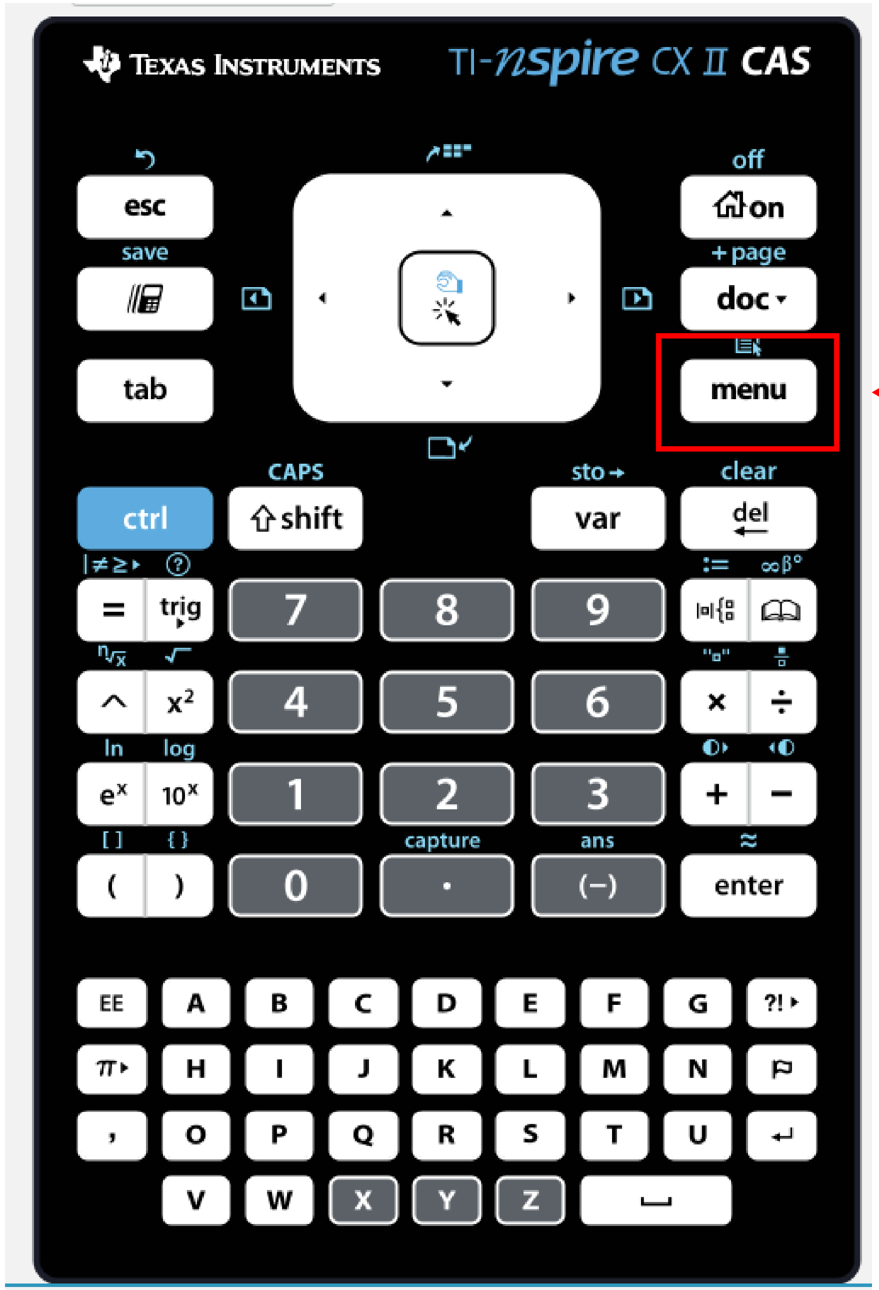
```
32   3.141592653701082
33   3.141592653584667
34   3.141592653642874
35   3.14159265361377
36   3.141592653599218
37   3.141592653591943
38   3.141592653588305
39   3.141592653590124
40   3.141592653589214
41   3.141592653589669
42   3.141592653589896
```

Python Shell 55/63

```
43   3.141592653589782
44   3.141592653589839
45   3.141592653589811
46   3.141592653589797
47   3.14159265358979
48   3.141592653589793
49   3.141592653589795
50   3.141592653589794
51   3.141592653589794
52   3.141592653589793
53   3.141592653589793
```

Python Shell 63/63

```
51   3.141592653589794
52   3.141592653589793
53   3.141592653589793
54   3.141592653589793
55   3.141592653589793
56   3.141592653589793
57   3.141592653589793
58   3.141592653589793
59   3.141592653589793
60   3.141592653589793
>>>
```

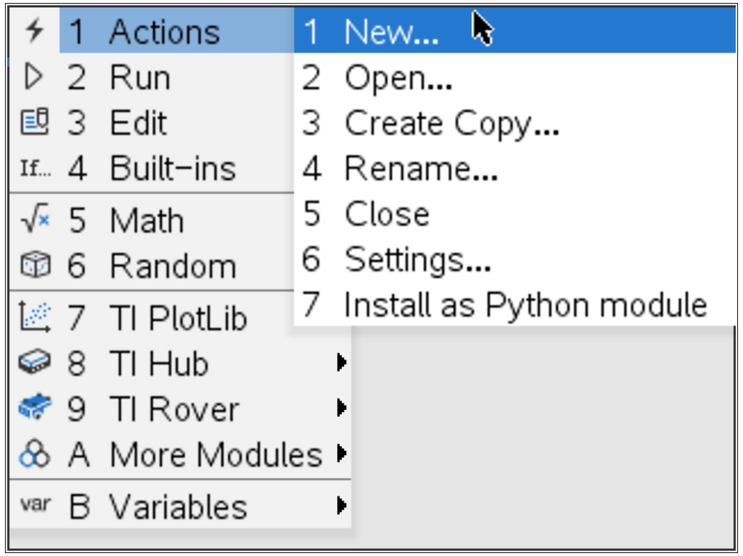


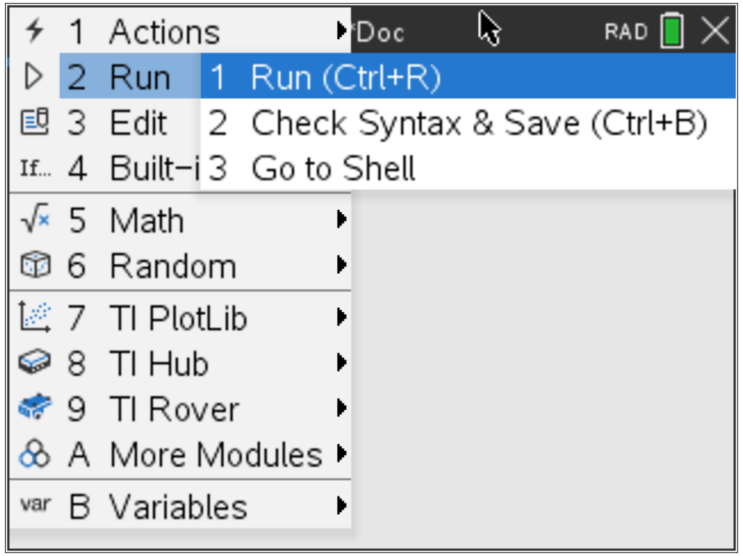
Python on the TI CAS has many of the commands built into its menu system.

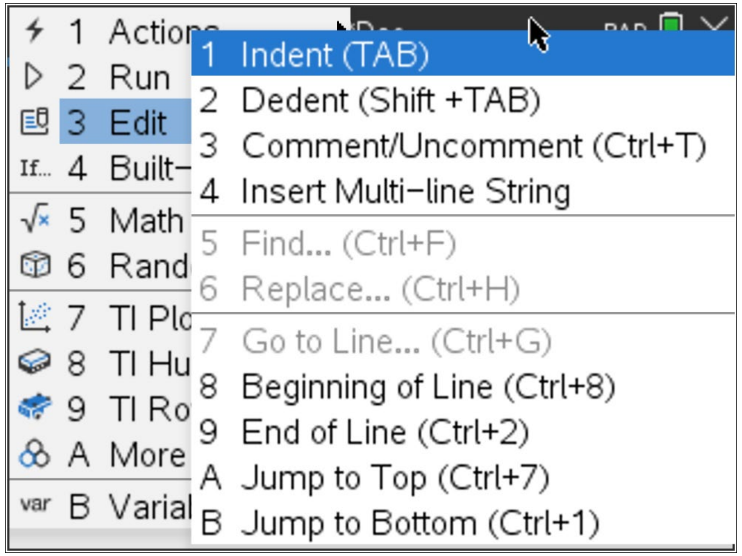
This will save you a lot of typing on the TI-CAS's non-QWERTY.

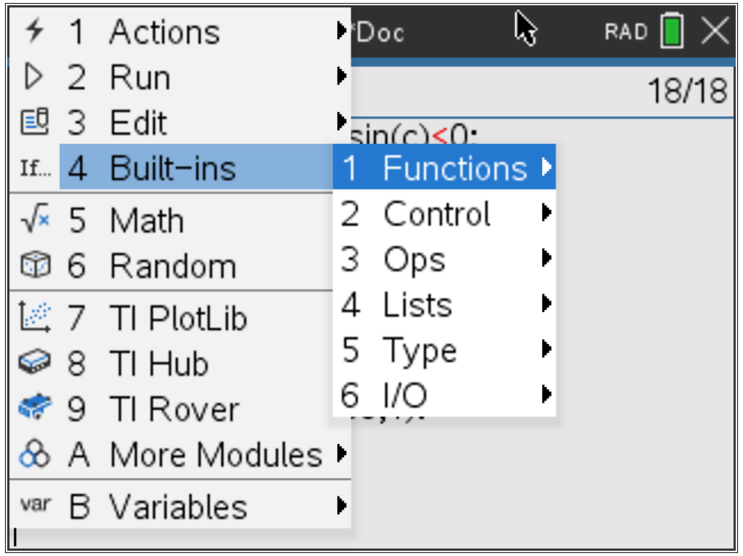
The following screens show the main Python menu features.

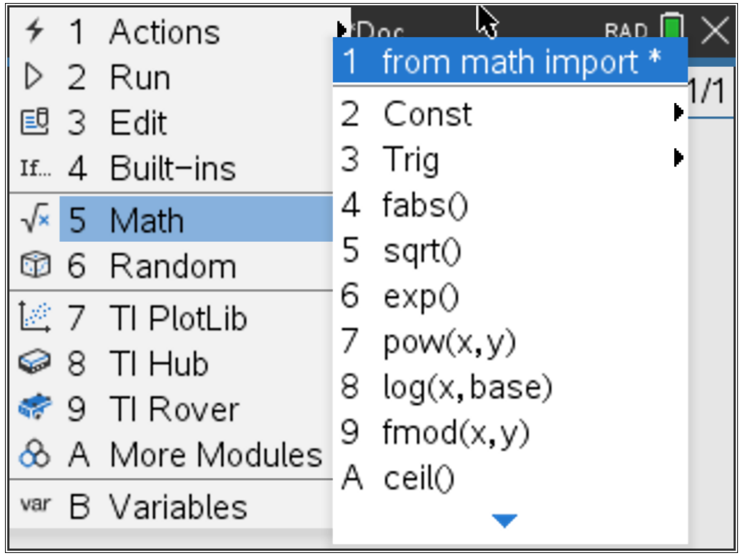
menu 1 2 3 4 5 6

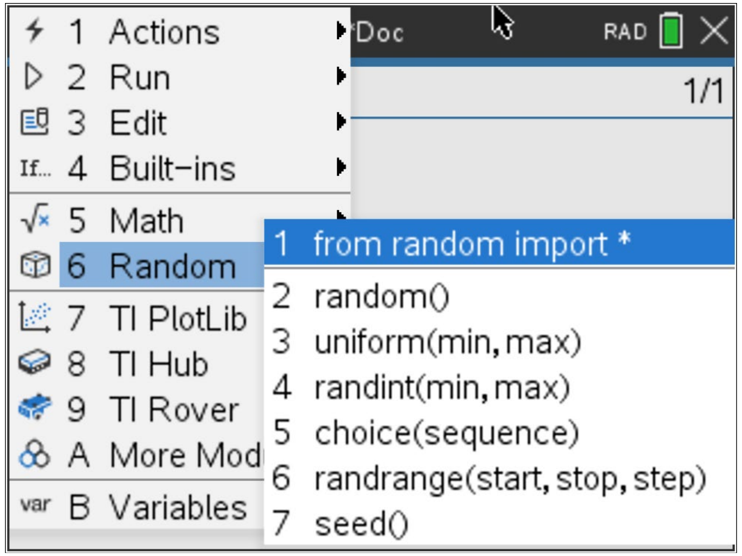

1 Actions
2 Run
3 Edit
4 Built−ins
5 Math
6 Random
7 TI PlotLib
8 TI Hub
9 TI Rover
A More Modules
B Variables
1 New...
2 Open...
3 Create Copy...
4 Rename...
5 Close
6 Settings...
7 Install as Python module


1 Actions
2 Run
3 Edit
4 Built−i
5 Math
6 Random
7 TI PlotLib
8 TI Hub
9 TI Rover
A More Modules
B Variables
Doc
RAD
1 Run (Ctrl+R)
2 Check Syntax & Save (Ctrl+B)
3 Go to Shell

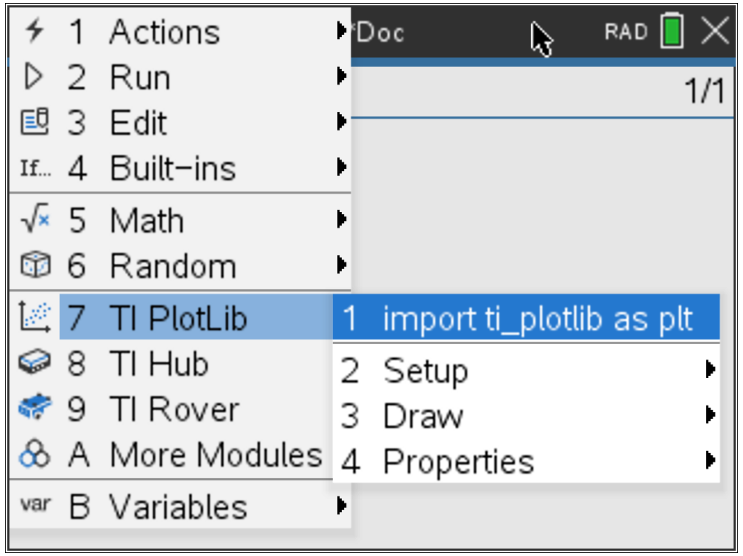

1 Indent (TAB)
2 Dedent (Shift +TAB)
3 Comment/Uncomment (Ctrl+T)
4 Insert Multi−line String
5 Find... (Ctrl+F)
6 Replace... (Ctrl+H)
7 Go to Line... (Ctrl+G)
8 Beginning of Line (Ctrl+8)
9 End of Line (Ctrl+2)
A Jump to Top (Ctrl+7)
B Jump to Bottom (Ctrl+1)

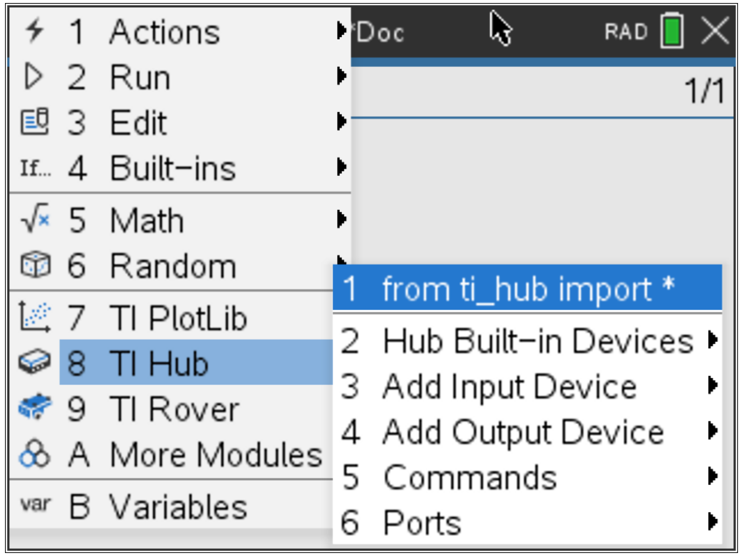

1 Actions
2 Run
3 Edit
4 Built−ins
5 Math
6 Random
7 TI PlotLib
8 TI Hub
9 TI Rover
A More Modules
B Variables
Doc
RAD
18/18
1 Functions
2 Control
3 Ops
4 Lists
5 Type
6 I/O

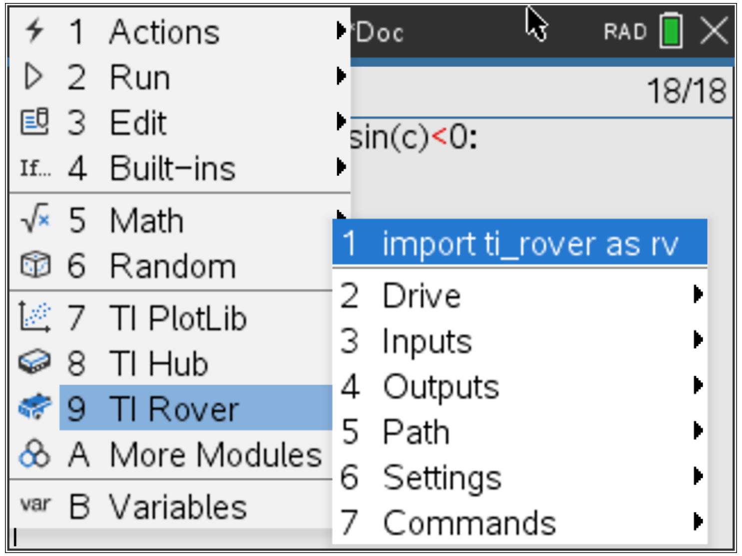

1 Actions
2 Run
3 Edit
4 Built−ins
5 Math
6 Random
7 TI PlotLib
8 TI Hub
9 TI Rover
A More Modules
B Variables
1 from math import *
2 Const
3 Trig
4 fabs()
5 sqrt()
6 exp()
7 pow(x,y)
8 log(x,base)
9 fmod(x,y)
A ceil()

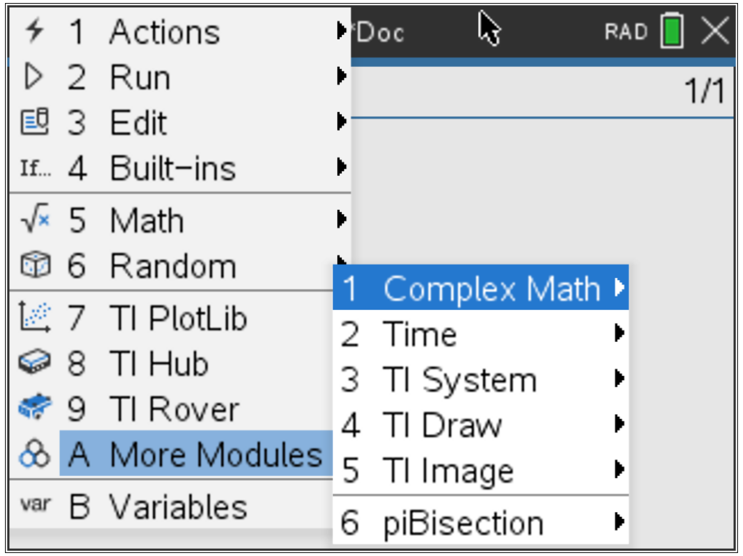

1 Actions
2 Run
3 Edit
4 Built−ins
5 Math
6 Random
7 TI PlotLib
8 TI Hub
9 TI Rover
B Variables
Doc
RAD
1/1
1 from random import *
2 random()
3 uniform(min, max)
4 randint(min, max)
5 choice(sequence)
6 randrange(start, stop, step)
7 seed()

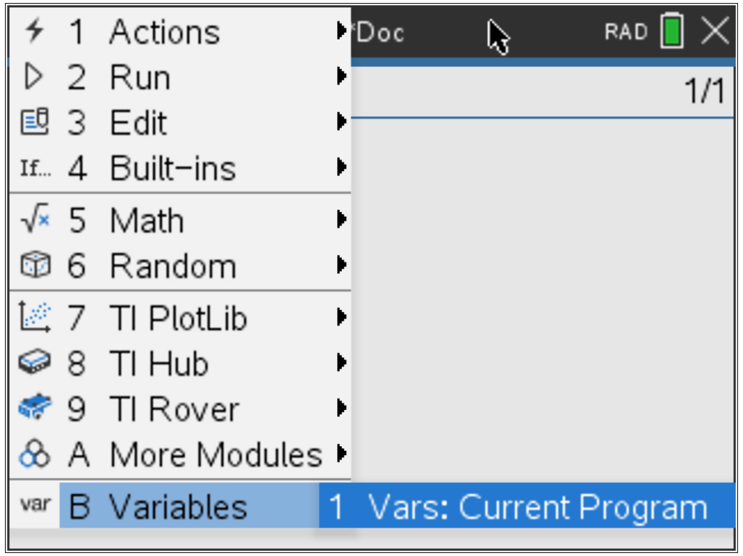
menu 7 8 9 A B











To stop an **infinite loop** in Python on the CAS, press and hold the **On** button for a few seconds. This will force it to stop. However, this will not work on the software emulator.

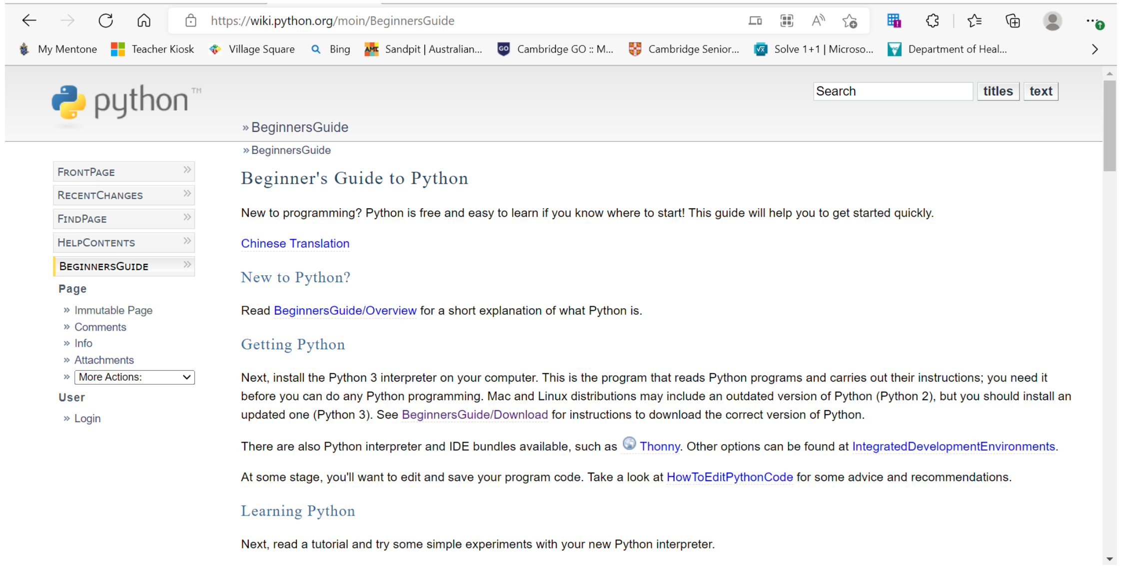
Part 5 of 5:

Free online resources:
Tutorials and
Integrated Developments Environments (IDEs)

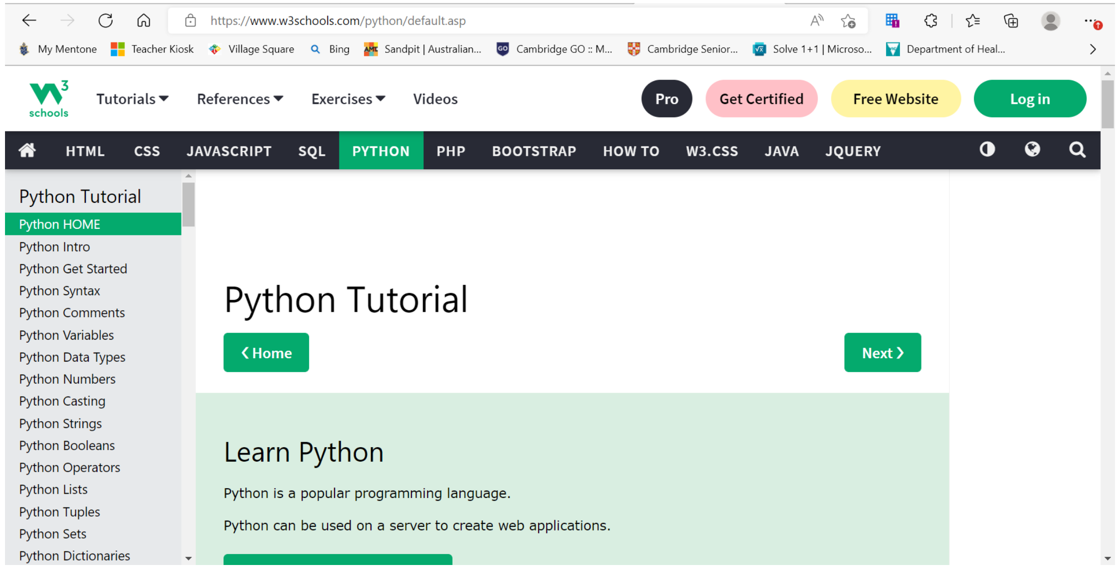
An IDE is where you can write and run your own
Python code to test your pseudocode and algorithms.

Free online tutorials for learning Python

https://wiki.python.org/moin/BeginnersGuide



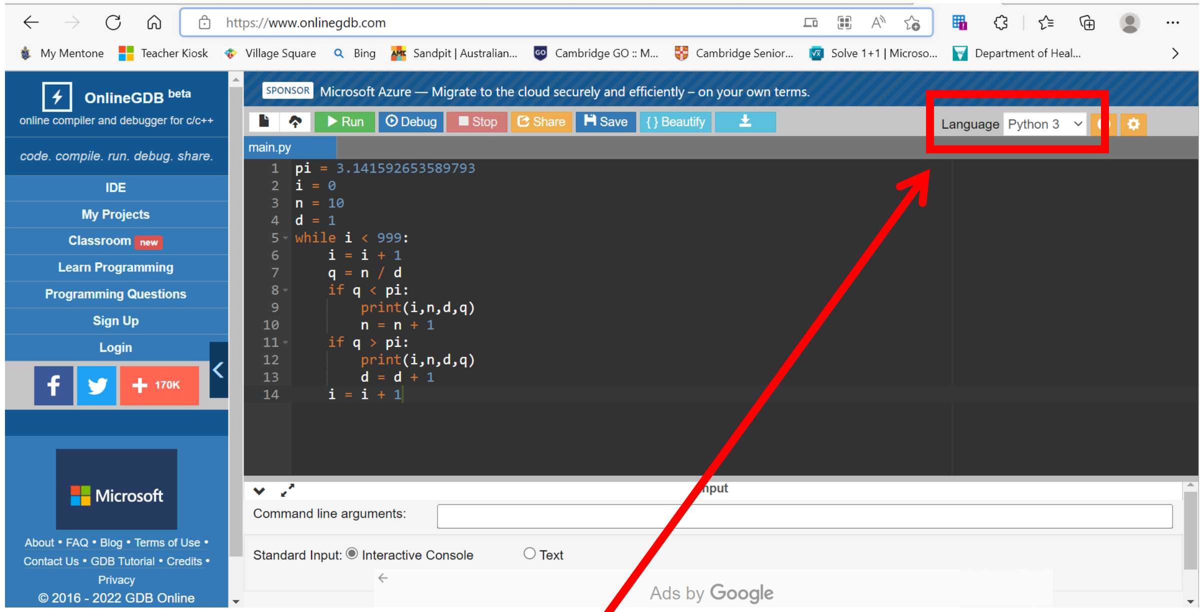
https://www.w3schools.com/python/default.asp

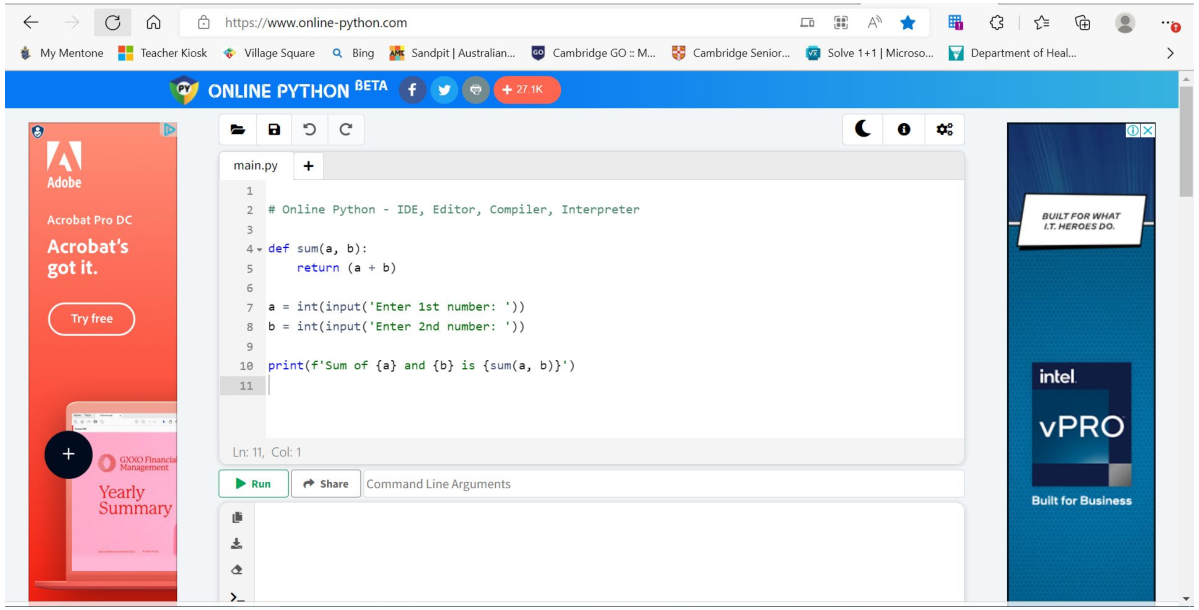


Free online Python Integrated Development Environments (IDEs).

https://www.onlinegdb.com/

https://www.online-python.com/





Make sure you select Python 3

Enzo Vozzo

After working as a Technical Officer at Telstra, Enzo graduated from Monash University in 2005 with a Bachelor of Technology (Computer Studies) and taught Electronics and Communications Engineering at Chisolm TAFE.

In 2013 he graduated from RMIT University with a Graduate Diploma of Education teaching Secondary School Mathematics and Science.

Since 2016 he has been teaching Mathematics at Mentone Grammar.

Email: exv@mentonegrammar.net

Instagram iphiepi: https://www.instagram.com/iphiepi/ →

$$i = \sqrt{-1} \qquad \phi = \frac{1+\sqrt{5}}{2}$$

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} \qquad \pi = 4\int_{0}^{1} \sqrt{1-x^2}\,dx$$

YouTube Channel: Maths Whenever:
https://www.youtube.com/channel/UCFLdfe_y2OQ1MZvGjha9taQ/videos